Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks

YOUCHENG SUN, University of Oxford, UK
MARCO DI NATALE, Scuola Superiore Sant'Anna, Pisa, Italy

The hard deadline model is very popular in real-time research, but is representative or applicable to a small number of systems. Many applications, including control systems, are capable of tolerating occasional deadline misses, but are seriously compromised by a repeating pattern of late terminations. The weakly hard real-time model tries to capture these requirements by analyzing the conditions that guarantee that a maximum number of deadlines can be possibly missed in any set of consecutive activations. We provide a new weakly hard schedulability analysis method that applies to constrained-deadline periodic real-time systems scheduled with fixed priority and without knowledge of the task activation offsets. The analysis is based on a Mixed Integer Linear Programming (MILP) problem formulation; it is very general and can be adapted to include the consideration of resource sharing and activation jitter. A set of experiments conducted on an automotive engine control application and randomly generated tasksets show the applicability and accuracy of the proposed technique.

CCS Concepts: • **Computer systems organization** → **Embedded software**; **Real-time operating systems**;

Additional Key Words and Phrases: Weakly hard real-time systems, schedulability analysis, periodic real-time tasks, resource sharing, activation jitter

ACM Reference format:
Youcheng Sun and Marco Di Natale. 2017. Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 171 (September 2017), 19 pages.
https://doi.org/10.1145/3126497

1 INTRODUCTION

The notion of a real-time task abstracts the execution of a program (e.g., a thread) triggered repeatedly by a (periodic) clock or a generic event, with a set of applied temporal constraints. Given a model of execution for a set of tasks and a scheduler, *hard* real-time schedulability analysis verifies if a system, that is, all of its tasks, can be safely guaranteed to complete at every activation before a deadline. The hard schedulability problem has been thoroughly investigated in the last decades and solved for many cases of interest.

This article was presented in the International Conference on Embedded Software 2017 and appears as part of the ESWEEK-TECS special issue.

The project leading to this application has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644080.

Authors' addresses: Y. Sun, University of Oxford, Wolfson Building, Parks Road, OXFORD, OX1 3QD, UK; email: youcheng.sun@cs.ox.ac.uk; M. Di Natale, Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56124 Pisa, Italy; email: marco@sssup.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2017 ACM 1539-9087/2017/09-ART171 $15.00
https://doi.org/10.1145/3126497

However, many systems, including control systems, are tolerant to individual deadline misses and treating them as hard real-time would result in unnecessary pessimism and possibly over-provisioning of resources. Of course, uncontrolled response times are also not desirable and even in case of deadline misses the designer may require some guarantees on the timing behavior of the system.

Among possible options, the *weakly hard* scheduling model has been proposed by several authors (Hamdaoui et al. [24] and Bernat et al. [6]) to check for a relaxed condition, that is, to analyze the number of temporal constraints violations given a time window or a sequence of task activations. This is also called *m-K* model, since a typical formulation consists in checking that no more than m deadlines are missed over a set of K activations. The rationale behind this model is that each deadline miss will bring the system closer to a faulty condition or state, and each completion in time will move the system back towards a safe working state.

The weakly hard model can be analyzed under several scheduling options, but due to its simplicity and effectiveness, fixed priority scheduling is nowadays the de facto standard for industrial real-time systems. This paper focuses on real-time systems consisting of periodic tasks scheduled with fixed priority.

The main problem with existing analysis methods is that for a weakly hard system, as for any system that can be (at least temporarily) overloaded, the critical instant theorem does not hold and the system becomes much more difficult to analyze. In the original work, this limitation has been overcome by restricting the study to offset-determined systems, that is, systems in which tasks are scheduled with a known initial offset.

This is a possibly serious limitation, not only because the designer may not be able to enforce or determine the system initial offsets, but especially because the analysis of all offset-determined systems is very sensitive to time drifts and errors in the task activation times, which are extremely hard to avoid in real systems.

In this work, we provide a generalized framework for offset-free systems scheduled on uniprocessors. Our formulation is based on a MILP encoding and the problem of the critical instant determination is addressed by letting a variable represent the beginning of the busy period. The MILP formulation can be easily reused to check the system for a large set of m and K values, allowing the designer to explore a wide design range. The developed MILP model serves as an over-approximate analysis, that is, if our weakly hard analysis confirms the *m-K* property, it is guaranteed that there will be no more than m deadline misses out of any K successive job activations (of the target task), however, if the *m-K* property is not confirmed, we cannot conclude the opposite.

Main contributions:

- We propose the first weakly hard schedulability analysis for offset-free periodic real-time tasks. The analysis method includes the consideration of resource sharing and activation jitter.
- To solve the possible issues with the large number of integer variables counting the number of task interferences (as used, for example, in [7, 19, 32]), we relaxed these variables to real values, but we added binary variables expressing the possibility of job interferences for reducing or possibly eliminating the introduced pessimism.
- Surprisingly, there is no existing work that can cope with the weakly hard analysis of general (offset-free) periodic tasks, and this prevents a fair comparison between our solution and other relevant works. Thus, we evaluate our analysis method through extensive experiments to show its efficiency (expected runtime) and precision. In the special case in which $m = 1$, the analysis is always accurate as long as it validates the *m-K* property. With

respect to accuracy, despite the relaxation to real valued job interference counters, the MILP analysis can still return exact results for a very high percentage of the tests.

Paper Organization: Section 2 introduces background results and possible applications that are related with the scheduling in overload conditions, the weakly hard real-time model and its analysis techniques. Next, we formally define the system model and the weakly hard analysis problem in Section 3. Section 4 contains the description of our proposed solution to the problem, which consists of a Mixed Integer Linear Programming (MILP) formulation. As a demonstration of the generality of the solution model, it is then extended in Section 5 to systems having mutual exclusive shared resources, and to tasks with jitter. The evaluation of the proposed technique is conducted in Section 6, and at last, Section 7 contains the conclusions and addresses future issues.

2 STATE OF THE ART

Since the seminal work of Liu and Layland [23], an overwhelming effort in real-time scheduling research has been dedicated to the question on whether there can be *a possible deadline miss* according to the *hard* real-time model. It is hard to completely identify the reasons for this disproportionate interest in hard analysis techniques. It is probably because of the simplicity of the model, its easier understandability and analyzability, the seemingly natural fit to safety-critical systems and, quite possibly, some incorrect judgement on the part of some researchers that believe most real-world systems are of hard-type. The success of hard schedulability analysis also benefits from the existence of the critical instant, an activation scenario where all tasks are simultaneously activated, that leads to the worst-case response time for every task in the system. As a result, in a hard real-time system it is sufficient to investigate the particular task activation pattern that originates from the critical instant. More details on classic hard real-time schedulability analysis can be found in textbooks and surveys like [12] and [28].

A simplistic version of the periodic task model assumes that the activation time of the first instance, or initial offset, of a task is known. For a system of periodic tasks with explicit initial offsets, Leung and Whitehead [22] proved that, starting from any job activation, it is necessary and sufficient to simulate the worst-case execution of the tasks in a bounded time interval to check if there is any deadline miss in the system, as the schedule of periodic tasks will repeat itself. However, as explained in [5], the result of this test is very sensitive to task parameters, including the initial offset.

While it is true that some safety-critical systems are vulnerable to a single violation of the temporal constraints, there are many more that can tolerate timing violations. In these cases, the hard schedulability analysis is too strict. The *weakly* hard real-time schedulability analysis targets the problem of *bounding the maximum number of deadline misses over a number of task activations.* A dynamic priority assignment of priority for streams with *m-K* requirements is proposed by Hamdaoui et al. [24] to reduce the probability of *m-K* violations in time-sensitive applications. Weakly hard real-time schedulability analysis can be traced back to the work of Bernat et al. [6] on the *m-K* model, in which no more than m deadline misses shall occur for any K consecutive activations of a task. The analysis in [6] and in other works assumes that there is an explicit initial state of the system, in which the initial offset of each task in the system is known. By restricting the analysis to a periodic activation pattern, the weakly hard analysis can be conducted by checking task activations and interleavings within a large enough time span from the system initialization, so as to verify the *m-K* assumption. Periodic tasksets are quite common in real applications, but the requirement of knowing all activation offsets may be too strict and undermine the robustness of the analysis: given a periodic task system with explicit initial offsets that passes the (weakly) hard test, a slightly change of the initial offset of some task may result in an unexpected time failure. The analysis is also very sensitive to a drift of the task periods.

Recent developments in the study of overloaded systems allow to relax the requirement of knowing the initial system state. The approach consists in the *worst-case analysis* [25] of a system model represented as the superposition of a typical behavior (e.g., of periodic task activations) that is assumed feasible, and a sporadic overload (i.e., a rare event). Under such an assumption, [18] and [33] proposed methods for weakly hard analysis that is composed by two phases: 1) the system is verified to be schedulable under the typical scenario (by the classical hard analysis), and 2) when the system is overloaded, it can be guaranteed that out of K successive activations of a task, at most m of them will miss the deadline. The sporadic behavior can be abstracted by observing and analyzing the system at runtime, and is characterized as a rare event. A similar approach is considered in [20], where real-time calculus is used to analyze the worst-case busy period (in duration and lateness) that results from a temporary overload because of an error in the timing assumptions on the task worst-case execution times. Both methods require the definition of a task model that may be artificial, since it requires the identification and separation of possible causes of overload. Finally, at least in principle, probabilistic analysis of deadline misses, such as the analysis in [14] could be used to compute the probability of missing m deadlines over K instances, but the model in [14] also assumes known activation offsets and is likely to be computationally extremely expensive when applied to the m-K analysis.

The analysis of overload conditions is also closely related to the co-design of control and CPU-time scheduling [4]. The influence of response times on the performance of control tasks has been studied in several works such as [34] and [35]. [3] proposed an integrated approach for controller synthesis, selecting task parameters that meet the expected control performance and guarantee the stability. [2] extended the idea in [3] from uniprocessors to distributed cyber physical systems, and in [17] FlexRay is considered as the communication medium. The m-K model has also been investigated in the co-design of controls (with respect to their performance) and scheduling [16, 26], and is used in [10, 11, 30, 31] to define the maximum number of samples (jobs) that can be dropped over any number of consecutive samples (density of dropped samples), to guarantee a minimum level of quality to the controls. However, [11] and [10] do not provide methods for the schedulability analysis of a system and the density of dropped samples needs to be enforced by the operating system or smart sensors.

3 THE SYSTEM MODEL

A periodic real-time task is characterized by a tuple $\tau_i = (C_i, D_i, T_i)$ with $C_i \leq D_i \leq T_i$ such that C_i is its Worst-Case Execution Time (WCET), D_i is the relative deadline and T_i is the period for the activation of τ_i. A task utilization is defined as $U_i = \frac{C_i}{T_i}$.

Each activation (instance) of τ_i is denoted by a job $J_{i,k}$ with $k = 1, 2, \ldots$ representing the job index (of its activations in time). A job $J_{i,k}$ can be further represented by its activation (or arrival) time $a_{i,k}$, its absolute deadline $d_{i,k} = a_{i,k} + D_i$, and its finish time $f_{i,k}$.

A job is schedulable if it can finish its execution before its deadline, i.e., $f_{i,k} \leq d_{i,k}$; and a task τ_i is schedulable if all its jobs are schedulable. The elapsed time between a job finish time and its activation time (i.e., $f_{i,k} - a_{i,k}$) is its *response time*. By definition, a task τ_i is schedulable if and only if the Worst-Case Response Time (WCRT) $R_i = \max_k\{f_{i,k} - a_{i,k}\}$ among all its jobs is not larger than its relative deadline D_i. A task Best-Case Response Time (BCRT) $r_i = \min_k\{f_{i,k} - a_{i,k}\}$ is the minimum time that it takes to complete the execution of the task.

In the periodic activation pattern $a_{k+1} - a_k = T_i$ for any two successive jobs of a task. As we do not require a specific initial offset for a task, the first job activation time $a_{i,1}$ of τ_i is unknown. A job (and the corresponding task) is said to be *active* if it has been activated but has not completed its execution.

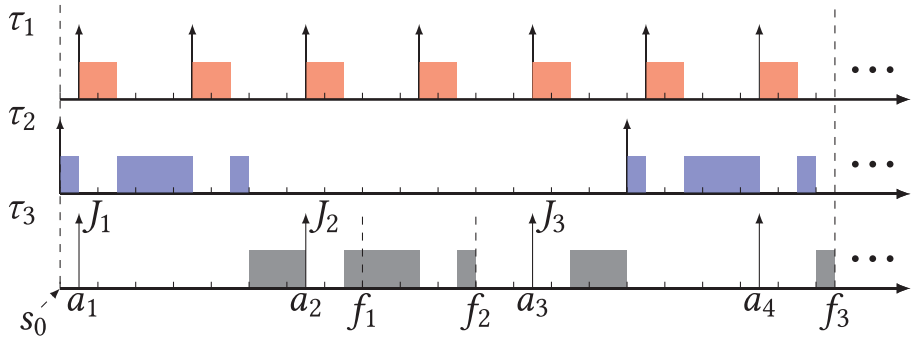


Fig. 1. A problem window with 3 job windows.

The periodic taskset $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ executes upon a uniprocessor platform. Each task in $\mathcal{T}$ is assigned a unique and static priority, and tasks are scheduled by a *fixed priority preemptive scheduler*. Tasks are ordered in $\mathcal{T}$ from higher to lower priority. That is, τ_j has a higher priority than τ_i, if $j < i$. If a task does not always finish before its deadline, it can have multiple active jobs at the same time. In such cases, these jobs are served in FIFO order.

A *level-i busy period* is defined as a time interval during which the processor is always occupied by the execution of tasks with priority higher than or equal to τ_i. For example, in Figure 1, $[s_0, f_2[$ and $[a_3, f_3[$ are level-3 busy periods. Because the focus of this paper is not computing the task WCRT, but analyzing all possible windows with missed deadlines, the definition of busy period extends the maximal level-i busy period as defined in [21].

The execution of any job of τ_i can only be affected by the workload of interfering tasks (including τ_i itself) within the same level-i busy period. According to [21], the WCRT R_i of a task τ_i is found within the longest level-i busy period, which starts at the critical instant (i.e., when all tasks are activated at the same time). In case a task always completes before its next activation, the task schedulability can be easily checked by computing the response time of the first task instance inside it.

However, this condition does not hold for weakly hard real-time systems when deadlines can be missed and multiple instances can be active at the same time. In this case, the WCRT of a task τ_i does not necessarily happen for the first job in a level-i busy period. However, the BCRT is still occurring for the last job in a level-i busy period. Algorithms to compute the BCRT can be found in [9, 27]. In this work, we trivially assume that the BCRT of a task does not exceed its period: $r_i \leq T_i$. Otherwise, the task simply misses all its deadlines. Also, we assume that the BCRT r_i and the WCRT R_i of each task are computed in advance using established techniques such as in [9, 21, 27]. Once computed, these values can be used as parameters in the MILP formulation.

We assume the system utilization $U = \sum_{1 \leq i \leq n} U_i$ is lower than 1, meaning that each job is guaranteed to complete its requested execution at some point in time, whether it misses its deadline or not. In addition, we use the result in [5] that shows that fixed priority scheduling analysis is sustainable, that is, if a job is not schedulable, then it will not become schedulable by increasing its execution time. From the analysis point of view, we simply assume that a task requests its WCET every time it is activated.

3.1 The Weakly Hard Model

This subsection formalizes the problem of weakly hard schedulability analysis. The analysis applies to an arbitrary task $\tau_i \in \mathcal{T}$, also defined as *target task*. An arbitrary sequence of K successive activations of τ_i is considered, with the objective of checking whether there are more than m deadline misses for τ_i in this sequence.

For simplicity, the jobs of τ_i in the activation sequence are denoted by $J_1, \ldots, J_k, \ldots, J_K$ (without the task index). Given a job J_k, its activation time and finish time are defined as a_k and f_k, respectively. The time interval $[a_k, a_{k+1}[$ is called the *kth job window* of τ_i, and the *problem window* for the analysis is $[s_0, f_K[$, where s_0 (also considered as the time reference $s_0 = 0$) is the earliest time instant such that the processor is fully occupied by the execution of higher priority tasks from s_0 to a_1.

As an example, consider a system of 3 tasks: $(C_1 = 1, D_1 = 3, T_1 = 3)$, $(C_2 = 3, D_2 = 5, T_2 = 15)$ and $(C_3 = 2, D_3 = 6, T_3 = 6)$, with $K = 3$ 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. In this case, $s_0 = 0$ and $a_1 = 0.5$. If the problem window starts at the critical instant, that is, when all tasks are synchronously activated in s_0, only J_1 misses its deadline.

4 THE SOLUTION MODEL

In this section we introduce the Mixed Integer Linear Programming (MILP) formulation for the weakly hard analysis of a set of offset-free periodic tasks under fixed priority scheduling. Two observations allow to reduce the problem space by considering only the problem windows that maximize the number of deadline misses for τ_i.

(**O1**) The worst-case number of deadline misses occurs for problem windows such that the last job of τ_i before the beginning of the problem window (indicated as J_0) is schedulable.

If J_0 is not schedulable, any problem window of K instances starting with J_0 has at least as many misses as the window starting with the first job J_1. The two windows have all the jobs from J_1 to J_{K-1} in common, but J_0 misses its deadline, therefore, in the best case the two windows have the same number of misses if also J_K is a deadline miss. In Figure 1, consider the problem window with J_2, J_3 and the following instance (not shown) J_4. Depending on the schedulability of J_4, there will be 1 or 2 deadline misses in this window. However, since J_1 is non-schedulable, there are 2 deadline misses for the problem window including J_1, J_2 and J_3.

(**O2**) The worst-case number of deadline misses occurs for problem windows such that the first job is non-schedulable.

Consider a window of K instances that starts with a set of schedulable jobs of arbitrary length $J_1, \ldots J_n$ (with $n < K$; if $n = K$ the proof is trivial) and the window that starts with J_{n+1}; the latter has at least as many deadlines misses as the window starting with J_1 (the proof is similar to the previous case).

4.1 Variables and Basic Constraints

In this subsection, we introduce the (real and Boolean) variables defined in our MILP model, together with some basic constraints on them. Real valued variables are labeled by $\mathbb{R}$ and Boolean variables by $\mathbb{B}$. M is a big enough constant value used to encode conditional constraints (a standard technique known as big-M). A brief summary of all the optimization variables is in Table 1.

Busy periods. Each job of τ_i inside the problem window can be interfered by pending executions of higher priority tasks that are requested before its activation but have not completed. The real valued L_k (as in Figure 2) indicates the portion of the level-i busy period for job J_k that extends to the earliest such activation, when $f_{k-1} \leq a_k$. That is, if J_{k-1} finishes execution not later than a_k (i.e., J_{k-1} does not interfere with the execution of J_k), $a_k - L_k$ is the earliest time instant such that, from $a_k - L_k$ to a_k, the processor is fully occupied by higher priority tasks.

The start time $s_0 = 0$ of our problem window is $a_1 - L_1$, and the arrival time a_k of the kth job of τ_i in the problem window is $a_k = L_1 + (k-1) \cdot T_i$. A trivial constraint that applies to all L_k is

$$\forall k \quad 0 \leq L_k \leq T_i - r_i \tag{1}$$

Table 1. A Summary of Variables Defined

Type	Variables	Annotations
$\mathbb{R}$	L_k	Segment of busy execution of higher priority tasks in front of (before) a job window, when $f_{k-1} \leq a_k$.
	α_j	Activation time of the 1st job of each higher priority task τ_i with respect to the start time s_0 of the analysis.
	f_k	Finish time of J_k.
	ι_k	Processor idle time inside the job window.
	$If_{j,k}$	#jobs of τ_j within the time interval $[0, f_k[$.
	$IL_{j,k}$	#job of τ_j inside $[0, a_k - L_k[$.
$\mathbb{B}$	b_k	$b_k = 0$ if J_k is schedulable; $b_k = 1$ if J_k misses its deadline. The number of deadline misses is $\sum_k b_k$.
	β_k	$\beta_k = 0$ if J_k completes before a_{k+1}; otherwise $\beta_k = 1$ and J_k interferes with J_{k+1}.
	$\Gamma f_{j,k}[\cdot]$	$\sum_p \Gamma f_{j,k}[p]$ is #jobs of τ_j inside: 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 $\beta_{k-1} = 1$.
	$\Gamma L_{j,k}[\cdot]$	$\sum_p \Gamma L_{j,k}[p]$ is #jobs of τ_j inside $[f_{k-1}, a_k - L_k[$, if it exists.

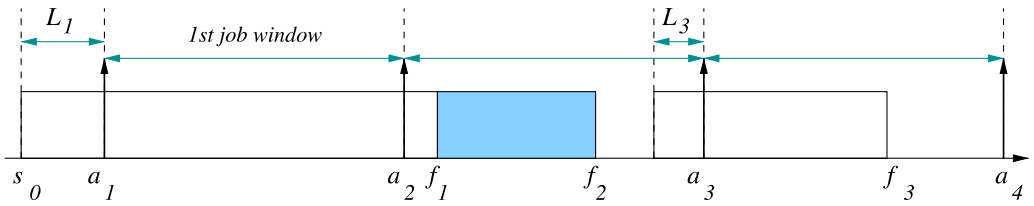


Fig. 2. Notation for the definition of a problem window.

Throughout our analysis, we only use the value of L_k, when $f_{k-1} \leq a_k$. If L_k is larger than $T_i - r_i$ then $f_{k-1} > a_k$ (any job of τ_i needs at least r_i to complete). In Figure 1, $L_1 = 0.5$ and $L_3 = 0$. Because J_1 interferes with the execution of J_2, L_2 is not relevant to our analysis.

Offsets. The offset of a higher priority task within the problem window refers to its first job activation time with respect to the start time s_0. The first job activation that happens no earlier

than s_0 for each higher priority task τ_j is denoted by $\alpha_j \in \mathbb{R}$

$$\forall j < i \;\; 0 \leq \alpha_j \leq T_j - r_j \tag{2}$$

Assume that the first job $J_{j,1}$ of τ_j in the window arrives at time $s_0 + T_j - r_j + \epsilon$ with $\epsilon > 0$. This implies that the previous job $J_{j,0}$ is activated at time $s_0 - r_j + \epsilon$. Because any job of τ_j needs at least r_j time to finish, $J_{j,0}$ will be still active at time instant 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 for α_j is $T_j - r_j$.

Finish times. For each job J_k of τ_i, its finish time is denoted by $f_k \in \mathbb{R}$

$$\forall k \;\; r_i \leq f_k - a_k \leq R_i$$

Because jobs from the same task are executed sequentially, for any two consecutive jobs of τ_i, this precedence constraint is encoded as

$$\forall k \;\; C_i \leq f_{k+1} - f_k \leq R_i + (T_i - r_i) \tag{3}$$

Level-i idle time inside a job window. The level-i processor idle time refers to the time when the processor is not occupied for execution by τ_i or any other higher priority task (in a given time interval). Given an arbitrary job window $[a_k, a_{k+1}[$ of J_k, we define $\iota_k \in \mathbb{R}$ as the amount of processor idle time inside this kth job window

$$\forall k \;\; 0 \leq \iota_k \leq a_{k+1} - a_k - r_i$$

Schedulability of each job of τ_i. For each job J_k of τ_i inside the problem window, a Boolean variable $b_k \in \mathbb{B}$ indicates whether the job misses its deadline:

- $b_k = 0$ if J_k finishes its execution no later than its deadline;
- $b_k = 1$ otherwise.

The value of b_k is defined by the comparison between the finish time f_k of J_k and its absolute deadline $a_k + D_i$: $b_k = 0 \Leftrightarrow f_k \leq a_k + D_i$, which is encoded by the following linear constraint.

$$\forall k \quad -M \cdot b_k \leq a_k + D_i - f_k < M \cdot (1 - b_k) \tag{4}$$

Being M a very large value, the conditional constraint in (4) forces $b_k = 0$ if the job J_k meets its deadline (i.e., $f_k \leq a_k + D_i$) and $b_k = 1$ otherwise. As in observation **O2** at the beginning of Section 4, we require that J_1 misses its deadline, that is, $b_1 = 1$ (schedulable tasks can be ruled out by simply performing a traditional hard schedulability test in advance).

The *total number of deadline misses* of τ_i inside the problem window is denoted by $\sum_k b_k$.

Interference from the previous jobs of the same task. A job J_k of τ_i interferes with the execution of the next job J_{k+1} in case $f_k > a_{k+1}$. The Boolean variable β_k encodes this condition.

- $\beta_k = 0$ if J_k finishes its execution within its own job window;
- $\beta_k = 1$ if J_k completes after a_{k+1}.

Similarly as in (4), the constraint $\beta_k = 0 \Leftrightarrow f_k \leq a_{k+1}$ over β_k, f_k and a_{k+1} can be formulated as

$$\forall k \quad -M \cdot \beta_k \leq a_{k+1} - f_k < M \cdot (1 - \beta_k) \tag{5}$$

If there is idle processor time inside the job window $[a_k, a_{k+1}[$ of J_k, then J_k must terminate within its window and does not interfere with J_{k+1} (i.e., $\beta_k = 1 \Rightarrow \iota_k = 0$).

$$\forall k \;\; \iota_k \leq M \cdot (1 - \beta_k)$$

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

Table 2. The Counting of Higher Priority Jobs

	$j=1$			$j=2$		
	$k=1$	$k=2$	$k=3$	$k=1$	$k=2$	$k=3$
$If_{j,k}$	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

common approach (as in [7, 19, 32]) is to count the number of jobs from each higher priority task that interfere with execution of the task under analysis. Different from previous works, we explore the relaxation of this integer count to a real value. Table 2 summarizes the variables defined for counting the higher priority interferences for the example system in Figure 1

Given a job J_k of τ_i and a higher priority task τ_j, $If_{j,k}$ is the number of jobs of τ_j within the time interval $[0, f_k[$. By definition, $If_{j,k} = \lceil \frac{f_k - \alpha_j}{T_j} \rceil$ is an integer number. However, we relax the definition of $If_{j,k}$ allowing it to be a real value and we linearize the constraint on $If_{j,k}$ as (by the definition in [21])

$$\forall j < i \;\; 0 \leq If_{j,k} - \frac{f_k - \alpha_j}{T_j} < 1 \tag{6}$$

Moreover, we define $IL_{j,k} \in \mathbb{R}$ as the number of jobs of τ_j ($\forall j < i$) within the time interval $[0, a_k - L_k[$, when $\beta_{k-1} = 0$. In this case, If J_k is not interfered by its predecessor J_{k-1}, then the number of jobs from τ_j that interfere with the execution of J_k is $If_{j,k} - IL_{j,k}$. We remind that we only use the value of L_k, and thus the interval $[0, a_k - L_k[$, when $\beta_{k-1} = 0$.

Formally, if J_{k-1} completes before the activation of J_k (i.e., $\beta_{k-1} = 0$), then $IL_{j,k} = \lceil \frac{a_k - L_k - \alpha_j}{T_j} \rceil$. That is, $\beta_{k-1} = 0 \Rightarrow 0 \leq IL_{j,k} - \frac{a_k - L_k - \alpha_j}{T_j} < 1$. In other words, $\forall j < i$

$$-M \cdot \beta_{k-1} \leq IL_{j,k} - \frac{a_k - L_k - \alpha_j}{T_j} < 1 + M \cdot \beta_{k-1} \tag{7}$$

In case $k = 1$, by the definition of the starting time instant $s_0 = 0$, it must be $\forall j < i : IL_{j,1} = 0$.

For simplicity, when $\beta_{k-1} = 1$, we force $IL_{j,k} = If_{j,k-1}$:

$$\forall j < i \quad -M \cdot (1 - \beta_{k-1}) \leq IL_{j,k} - If_{j,k-1} \leq M \cdot (1 - \beta_{k-1}) \tag{8}$$

Refining the interferences from higher priority tasks. Both $If_{j,k}$ and $IL_{j,k}$ are real variables. This is efficient but inaccurate. To restore a correct formulation, we define two classes of Boolean variables to constrain the values of $If_{j,k}$ and $IL_{j,k}$.

Given a job J_k of τ_i and a higher priority task τ_j, an array of Boolean variables $\Gamma f_{j,k}[p] \in \mathbb{B}$ counts the number of jobs of τ_j inside the time interval below (p indexes these jobs).

- $[a_k - L_k, f_k[$ if J_{k-1} does not interfere with J_k, i.e., $\beta_{k-1} = 0$;
- $[f_{k-1}, f_k[$ if J_{k-1} does interfere with J_k, i.e., $\beta_{k-1} = 1$.

A rough bound for the size of $\Gamma f_{j,k}[\cdot]$ (the number of instances of τ_j in the interval) is

$$\left\lceil \frac{R_i + T_i - r_i}{T_j} \right\rceil \tag{9}$$

$\Gamma f_{j,k}[p] = 1$ indicates that the pth job activation of τ_j in the specified time interval can interfere with the execution of J_k (the pth job is activated before J_k completes, otherwise, $\Gamma f_{j,k}[p] = 0$). The

total number of activations of jobs of τ_j, interfering with the execution of J_k in the specified time interval is

$$\Delta_{j,k} := \sum_p \Gamma f_{j,k}[p]$$

As shown in Table 2, for the example in Figure 1, when $j = 1$ and $k = 3$ it is $L_3 = 0$ and $\Delta_{1,3} = 3$ jobs from τ_1 within the time interval $[a_3 - L_3, f_3[$. As $\beta_1 = 1$, from f_1 to f_2, $\Delta_{1,2} = 1$.

In case J_{k-1} does not delay the execution of J_k, it is $IL_{j,k} + \Delta_{j,k} = If_{j,k}$. In the other case (i.e., when $\beta_{k-1} = 1$) $If_{j,k-1} + \Delta_{j,k} = If_{j,k}$ and (8) enforces $IL_{j,k} = If_{j,k-1}$. Consequently,

$$\forall j < i \;\; IL_{j,k} + \Delta_{j,k} = If_{j,k} \tag{10}$$

If a higher priority job $J_{j,p}$ does not interfere with job J_k of the target task, this implies that J_k completes before $J_{j,p}$, then no later job $J_{j,p'}$ ($p' > p$) can interfere with J_k. This results in the precedence constraint between elements inside the array $\Gamma f_{j,k}$.

$$\forall j < i \;\; \Gamma f_{j,k}[p+1] \leq \Gamma f_{j,k}[p]$$

Similarly, given a job J_k and a higher priority task τ_j, we define an array of Boolean variables $\Gamma L_{j,k}[\cdot]$ to count the number of jobs of τ_j inside the time interval $[f_{k-1}, a_k - L_k[$. The size of $\Gamma L_{j,k}[\cdot]$ can also be bounded, e.g., by

$$\left\lceil \frac{T_i - r_i}{T_j} \right\rceil \tag{11}$$

and the total number can be computed as

$$\Lambda_{j,k} := \sum_p \Gamma L_{j,k}[p]$$

$\Lambda_{j,k}$ counts the number of jobs from a higher priority task τ_j that are guaranteed not to interfere with J_{k-1} or J_k since they are activated after J_{k-1} finishes and are not in the same busy period with J_k. For instance, during the interval $[f_2, a_3 - L_3[$ in Figure 1, no higher priority jobs are activated: $\Lambda_{1,3} = \Lambda_{2,3} = 0$.

When $\beta_{k-1} = 0$, it is $If_{j,k-1} + \Lambda_{j,k} = IL_{j,k}$:

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

In case $\beta_{k-1} = 1$, the interval $[f_{k-1}, a_k - L_k[$ is not relevant to our analysis and we force $\Lambda_{j,k} = 0$, using the big-M formulation.

$$\forall j < i \quad -M \cdot (1 - \beta_{k-1}) \leq \Gamma L_{j,k}[p] \leq M \cdot (1 - \beta_{k-1}) \tag{13}$$

The constraint between variables in $\Gamma L_{j,k}[\cdot]$ resulting from the execution in FIFO order of the jobs from the same task can be encoded as

$$\forall j < i \;\; \Gamma L_{j,k}[p+1] \leq \Gamma L_{j,k}[p]$$

4.2 Constraints on the Idle Time and Workload

In this subsection, we present constraints that bound the processor idle times and the time spent executing by the tasks (i.e., workload) inside the problem window and its sub parts (e.g., one or multiple job windows). For short, we first define several terms: $\rho_k = f_k - a_k$, $\lambda_k = f_k - (a_k - L_k)$ and $\lambda'_k = f_k - f_{k-1}$. As an example, in Figure 1, $\rho_1 = 7.5$, $\lambda_1 = 8$ and $\lambda'_2 = 3$.

Minimum level-i idle time. To analyze the target task τ_i under fixed priority scheduling, it is sufficient to consider the taskset composed by τ_i and its higher priority tasks: $\mathcal{T}_i = \{\tau_1, \ldots, \tau_i\}$. Given an arbitrary time interval of length X, we use $\mathsf{minIdle}(\mathcal{T}_i, X)$ to denote the minimum amount of (level-i) processor idle time that is available within it (left unused by the tasks in $\mathcal{T}_i$).

Then, for any number x of consecutive job windows inside the problem window (of length $x \cdot T_i$), the total amount of idle time is lower bounded by $\mathsf{minIdle}(\mathcal{T}_i, x \cdot T_i)$. That is, $\forall 1 \le x \le K, 1 \le y \le K - x + 1$:

$$\mathsf{minIdle}(\mathcal{T}_i, x \cdot T_i) \le \sum_{y \le k \le y+x-1} \iota_k \tag{C1}$$

To compute $\mathsf{minIdle}(\mathcal{T}_i, X)$, we define a virtual task $\tau_* = (-, X, X)$ that has relative deadline and period equal to the interval length X and lowest priority. $\mathsf{minIdle}(\mathcal{T}_i, X)$ is estimated as the maximum execution time C of τ_* that still guarantees its schedulability: if C is not the minimum level-i idle time, then there should exist a combination of job activations for tasks in $\mathcal{T}_i$ that leads to a deadline miss for τ_* (easily demonstrated by contradiction; slack stealing algorithms [13] provide methods to estimate the processor idle time).

Idle time inside a job window. Consider the job window $[a_k, a_{k+1}[$ of J_k, if $\beta_k = 0$, ι_k is in fact the idle time the interval $[f_k, a_{k+1} - L_{k+1}[$, as exemplified by the 2nd job window in Figure 1. The total amount of higher priority workload in $[f_k, a_{k+1} - L_{k+1}[$ can be represented as

$$\Theta_k := \sum_{j<i} (IL_{j,k+1} - If_{j,k}) \cdot C_j$$

As a result, the idle time in the kth job window is $\iota_k = (a_{k+1} - a_k) - \rho_k - L_{k+1} - \Theta_k$. This equivalence only applies when $\beta_k = 0$ and can be encoded in a MILP formulation with the following constraint (trivially true for $\beta_k = 1$).

$$\forall k \quad -M \cdot \beta_k \le \iota_k + \Theta_k - (a_{k+1} - a_k - \rho_k - L_{k+1}) \le M \cdot \beta_k \tag{C2}$$

Formulation of the busy period $[a_k - L_k, f_k[$ *when* $\beta_{k-1} = 0$. If $\beta_{k-1} = 0$, J_{k-1} does not interfere with the execution of J_k, and $[a_k - L_k, f_k[$ is a busy period with length λ_k. The total amount of workload from higher priority tasks inside $[a_k - L_k, f_k[$ is

$$\Phi_k := \sum_{j<i} (If_{j,k} - IL_{j,k}) \cdot C_j$$

For the first instance $k = 1$, it is

$$\Phi_1 + C_i - \lambda_1 = 0 \tag{C3}$$

Otherwise, $\beta_{k-1} = 0$ implies that $\Phi_k + C_i = \lambda_k$. To apply the constraint only to the case $\beta_{k-1} = 0$, the formulation is

$$\forall k \quad -M \cdot \beta_{k-1} \le \Phi_k + C_i - \lambda_k \le M \cdot \beta_{k-1} \tag{C4}$$

Formulation of the busy period $[f_{k-1}, f_k[$ *when* $\beta_{k-1} = 1$. If $\beta_{k-1} = 1$, the interval between f_{k-1} and f_k is a busy period with length λ'_k. The total amount of workload from higher priority tasks inside $[f_{k-1}, f_k[$ is

$$\Phi'_k := \sum_{j<i} (If_{j,k} - If_{j,k-1}) \cdot C_j$$

Thus, the length λ'_k of busy period $[f_{k-1}, f_k[$ can be represented as $\Phi'_k + C_i$ and the MILP constraint becomes

$$\forall k \quad -M \cdot (1 - \beta_{k-1}) \le \Phi'_k + C_i - \lambda'_k \le M \cdot (1 - \beta_{k-1}) \tag{C5}$$

Formulation of f_k by accumulating the idle time and workload. If we consider each job J_k, from $s_0 = 0$ to its finish time f_k, the time interval $[0, f_k[$ consists of multiple busy periods and processor idle times, which can be summed up as in

$$\forall k \quad \sum_{j<i} If_{j,k} \cdot C_j + k \cdot C_i + \sum_{k'<k} \iota_{k'} = f_k \tag{C6}$$

Refining the arrival time of a higher priority job before the beginning or the end of a busy period. At the beginning or the end of a level-i busy period, a higher priority task must have completed any previously requested execution time. As a result, it must be

$$\forall k, j < i \quad \alpha_j + (IL_{j,k} - 1) \cdot T_j + r_j - M \cdot \beta_{k-1} < a_k - L_k \tag{C7}$$

The latest activation time of a higher priority job (from τ_j) before the beginning of a busy period starting in $a_k - L_k$ is $\alpha_j + (IL_{j,k} - 1) \cdot T_j$. This job must complete before the start of the busy period in $a_k - L_k$ after at least r_j time units. The term $-M \cdot \beta_{k-1}$ is used in the constraint (C7) because L_k is only relevant when $\beta_{k-1} = 0$.

Likewise, at the end of a busy period

$$\forall k, j < i \quad \alpha_j + (If_{j,k} - 1) \cdot T_j + r_j < f_k \tag{C8}$$

Length of a busy period. We use BP to denote the length of longest level-i busy period, and $N_i = \lceil \frac{BP}{T_i} \rceil$ is the number of jobs of τ_i within that busy period. As long as there is a busy period that spans N_i jobs of τ_i, the total task execution within it cannot exceed BP. Therefore, $\forall 1 \leq x \leq K - N_i + 1$:

$$\begin{aligned} &L_x + (a_{x+1} - a_x) - M \cdot (1 - \beta_x) + \\ &\sum_{x<k<x+N_i-1} (a_{k+1} - a_k - M \cdot (1 - \beta_k)) + \rho_{x+N_i-1} \leq BP \end{aligned} \tag{C9}$$

For arbitrarily N_i successive jobs inide an arbitrary problem window, we do not know if they are inside the same busy period, however, $\beta_k = 1$ is a sufficient condition for two jobs J_k and J_{k+1} to be in the same busy period (the same for $\beta_x = 1$) and this explains the big-M terms in (C9). For the scenario in Figure 1 where $BP = 11$ and $N_3 = 2$, there is a busy period $[s_0, f_2[$ that spans two jobs J_1 and J_2: $L_1 + (a_2 - a_1) + \rho_2 \leq BP$.

4.3 Weakly Hard Schedulability Analysis

Given an arbitrary sequence of K successive jobs of τ_i inside the problem window, the weakly hard property specifies that the maximum number of deadline misses (among these K jobs) should be bounded by m ($< K$). The total number of deadline misses can be computed as

$$\text{nDmiss} := \sum_k b_k \tag{C10}$$

The number of deadline misses of τ_i within the problem window is bounded by m, *if the addition of the constraint*

$$m + 1 \leq \text{nDmiss} \tag{C11}$$

makes the formulation non feasible.

Another option is to use the formulation of the number of deadline misses in (C10) as an optimization (maximization) function, and check what is the maximum number of misses for a given number of activations, or even the other way around, to find what is the minimum value of K given a number of deadline misses.

5 EXTENSIONS OF THE SOLUTION MODEL

The weakly-hard analysis framework proposed in Section 4 can be easily adapted to a more general task model, in particular, to shared resources and tasks with jitter.

5.1 Shared Resources

In this part, we show an extension to the case of resource sharing using the Immediate Priority Ceiling Protocol (PCP) [29] as used in the OSEK and AUTOSAR operating system standards.

A set of shared resources $\mathcal{R}_1, \ldots, \mathcal{R}_G$ are accessed by tasks in mutual exclusive mode. For any task τ_i and for any resource $\mathcal{R}_g$, $\mathcal{S}_{i,g} = \{cs_{i,g,1}, cs_{i,j,2}, \ldots\}$ is a finite **multiset** (a set that allows multiple instances of its elements) of worst case execution times for the critical sections executed by τ_i on $\mathcal{R}_g$.

The *priority ceiling* $pc(\mathcal{R}_g) := \min\{i : \mathcal{S}_{i,g} \neq \emptyset\}$ of $\mathcal{R}_g$ is defined as the highest priority of any task that accesses it. Every time a task accesses $\mathcal{R}_g$, its priority is boosted to the priority ceiling of $\mathcal{R}_g$. In this way, any job of τ_i can be blocked at most once by one lower priority job executing a critical section on a resource with priority ceiling $pc(\mathcal{R}_g) \leq i$. This guarantees a predictable worst-case blocking time.

For simplicity, in the following, we will assume the Rate Monotonic (RM) system such that τ_i has a higher priority than τ_j if $T_i < T_j$ and for any τ_i, $D_i = T_i$.

An arbitrary sequence of x consecutive job activations of τ_i, can be (directly or indirectly) blocked by at most x critical section executions on resources with ceiling higher than or equal to the priority of the job.

$$\mathcal{S}_i^x := \bigcup_{pc(\mathcal{R}_g) \leq i} \bigcup_{i<j} \overbrace{\mathcal{S}_{j,g} \cup \cdots \cup \mathcal{S}_{j,g}}^{\lceil \frac{BP + x \cdot T_i}{T_j} \rceil \text{ times}}$$

Hence, for any x consecutive job windows of τ_i, the maximum blocking time is defined as the sum of the x largest elements in the multiset $\mathcal{S}_i^x$:

$$\mathcal{B}_i^x := \sum_{\text{the } x \text{ largest}} \mathcal{S}_i^x$$

To apply these blocking times to the MILP model in Section 4, we follow the common approach that adds blocking time to execution time when considering the possible interference.

For any $1 \leq k \leq K$, the real variable $c_{i,k}$ indicates the execution time which includes the blocking time that the kth job of τ_i, within the problem window, can suffer.

$$C_i \leq c_{i,k} \leq C_i + \mathcal{B}_i^1 \tag{14}$$

For any number of consecutive job windows, we can bound the sum of all these execution variables: $\forall 1 < x \leq K \; \forall 1 \leq y \leq K - x + 1$

$$\sum_{y \leq k \leq y+x-1} c_{i,k} - x \cdot C_i \leq \mathcal{B}_i^x \tag{C12}$$

To extend the original problem formulation to the case of resource sharing, all instances of C_i in constraints (C3)~(C6) should be replaced by the corresponding variable $c_{i,k}$ Also the definition of the minimum processor idle time needs to be modified and the constraint (C1) is then updated as follows

$$\text{minIdle}(\mathcal{T}_i, x \cdot T_i) - \mathcal{B}_i^x \leq \sum_{y \leq k \leq y+x-1} \iota_k \tag{C1*}$$

5.2 Jitter

The jitter [9] of a periodic task represents the maximum possible delay of task actual activation times with respect to the ideal periodic activations. Given a periodic task $\tau_l = (C_l, D_l, T_l)$, we denote its jitter as $\mathcal{J}_l$ with $\mathcal{J}_l + C_l \leq D_l$.

Because of jitter, the distance between activation times of two jobs J_k and J_{k+1} of the target task τ_i inside the problem window is not a fixed value T_i, but can be any value within the range $[T_i - \mathcal{J}_i, T_i + \mathcal{J}_i]$. More generally, there is $\forall 1 \leq k < K, 1 \leq N \leq K - k$,

$$N \cdot T_i - \mathcal{J}_i \leq a_{k+N} - a_k \leq N \cdot T_i + \mathcal{J}_i$$

The jitter of a higher priority task τ_j also affects its interference upon the task under analysis. For example, the number of jobs of τ_j that arrive before the finish time of the kth job of τ_i within the problem window becomes: $\lceil \frac{f_k - \alpha_j - \mathcal{J}_j}{T_j} \rceil \leq If_{j,k} \leq \lceil \frac{f_k - \alpha_j + \mathcal{J}_j}{T_j} \rceil$. This is encoded by the constraint below, as a replacement of (6).

$$\forall j < i \quad \frac{f_k - \alpha_j - \mathcal{J}_j}{T_j} \leq If_{j,k} < \frac{f_k - \alpha_j + \mathcal{J}_j}{T_j} + 1 \tag{15}$$

For $IL_{j,k}$, when $\beta_k = 0$, it is now $\lceil \frac{a_k - L_k - \alpha_j - \mathcal{J}_j}{T_j} \rceil \leq IL_{j,k} \leq \lceil \frac{a_k - L_k - \alpha_j + \mathcal{J}_j}{T_j} \rceil$, and the big-$M$ constraint in (7) is updated to $\forall j < i$

$$\frac{a_k - L_k - \alpha_j - \mathcal{J}_j}{T_j} - M \cdot \beta_{k-1} \leq IL_{j,k} \tag{16}$$

$$IL_{j,k} < \frac{a_k - L_k - \alpha_j + \mathcal{J}_j}{T_j} + 1 + M \cdot \beta_{k-1} \tag{17}$$

To take into account jitter, several equations in the MILP formulation also need to be updated (the jitter mostly result in a modifier applied to periods). Summarizing, Equations (1), (2), (3), (4), (9), (11), (C1) (C7), (C8) are replaced with the following

$$\forall k \quad 0 \leq L_k \leq T_i - r_i + \mathcal{J}_i \tag{18}$$

$$\forall j < i \quad 0 \leq \alpha_j \leq T_j - r_j + \mathcal{J}_j \tag{19}$$

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

$$\forall k \quad -M \cdot b_k \leq a_k + D_i - f_k - \mathcal{J}_i < M \cdot (1 - b_k) \tag{21}$$

$$\left\lceil \frac{R_i + T_i - r_i + \mathcal{J}_j}{T_j} \right\rceil \tag{22}$$

$$\left\lceil \frac{T_i - r_i + \mathcal{J}_j}{T_j} \right\rceil \tag{23}$$

$$\mathsf{minIdle}(\mathcal{T}_i, x \cdot T_i - \mathcal{J}_i) \leq \sum_{y \leq k \leq y+x-1} \iota_k \tag{24}$$

$$\forall k, j < i \quad \alpha_j + (IL_{j,k} - 1) \cdot T_j + r_j - M \cdot \beta_{k-1} - \mathcal{J}_j < a_k - L_k \tag{25}$$

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

Table 3. An Automotive Case Study

Task	C_i	T_i	Task	C_i	T_i
τ_1	1015.83	$2 \cdot 10^4$	τ_8	5296.84	$1 \cdot 10^5$
τ_2	2309.5	$2 \cdot 10^4$	τ_9	325.64	$2 \cdot 10^5$
τ_3	1148.64	$2.5 \cdot 10^4$	τ_{10}	3285.24	$2 \cdot 10^5$
τ_4	2419.6	$3 \cdot 10^4$	τ_{11}	208.67	$5 \cdot 10^5$
τ_5	287.5	$5 \cdot 10^4$	τ_{12}	539.5	$5 \cdot 10^5$
τ_6	51.072	$6 \cdot 10^4$	τ_{13}	47616.3	$1 \cdot 10^6$
τ_7	2318.42	$1 \cdot 10^5$	τ_{14}	799006	$2 \cdot 10^6$
			τ_{15}	$1.0005 \cdot 10^6$	$1 \cdot 10^7$

6 EXPERIMENTS

In the section, we apply the proposed weakly hard schedulability analysis to an automotive engine control application and a set of randomly generated system configurations. All experiments are conducted on a machine with 8 GB memory and 8 cores: Intel(R) Xeon(R) CPU X3460 @ 2.80GHz, using CPLEX 12.6.3 as the MILP solver. The MILP formulation is encoded in C++ using the CPLEX library and is available for download[1].

6.1 The Fuel Injection Case Study

At first, we apply the MILP weakly hard schedulability analysis with the shared resource extension (Section 5), to the fuel injection application described in [8].

According to the AUTOSAR standard, an automotive application is composed by a set of functions called runnables, which are executed by tasks scheduled by fixed priority. The runnable-to-task mapping and the task scheduling are defined at the system integration phase.

For the fuel injection application in [8], a heuristic strategy is applied to allocate approximately 1000 runnables to tasks with 280 critical sections. The resulting taskset has 15 tasks with priorities assigned according to the Rate Monotonic rule (all times in microseconds)

Due to the blocking from τ_{15}, τ_{14} is not (hard real-time) schedulable. To verify the weakly hard property, we tested a series of m-K parameters: $\{(1, 5), (2, 5), (2, 10), (3, 10), (3, 15), (4, 15)\}$. According to our weakly hard schedulability analysis, it is guaranteed that there will be at most $m = 2$ (resp. 3 and 4) deadline misses out of any $K = 5$ (resp. 10 and 15) consecutive jobs of τ_{14}.

Regarding the runtime cost, except for the case $m = 3$ and $K = 15$, all tests complete within 2 minutes. It takes the CPLEX solver almost 30 minutes to make a decision when $m = 3$ and $K = 15$.

6.2 Runtime Performance

In this subsection, we apply the weakly hard real-time analysis in Section 4 to a set of randomly generated tasksets for an empirical evaluation of the runtime performance, with a large variety of configurations: $n \in \{10 \sim 15, 20, 30, 50\}$, $U \in \{0.8, 0.85, 0.9, 0.95\}$, $m \in \{1, 2, 3\}$ and $K \in \{5, 10 \sim 15, 20\}$. Each configuration in the experiment is specified by a tuple (n, U, m, K), where n is the taskset size, U is the taskset utilization, and m-K is the weakly hard property to be checked.

Overall, 6253 task systems are tested. For each taskset with a pair n and U: 1) the utilization U_i of tasks is generated using the Randfixedsum algorithm in [15]; 2) the task period T_i is uniformly sampled in the range $[10, 1000]$; each task has an implicit deadline, i.e., $D_i = T_i$; 3) and each task WCET is computed as $C_i = T_i \cdot U_i$.

[1] https://github.com/m-k-wsa/.

Table 4. Percentage of Sets Without K Consecutive Deadline Misses

	$K=2$	$K=3$		$K=2$	$K=3$
$n=10$	80.3%	98.6%	$n=30$	85.1%	99.9%
$n=20$	84.8%	99.6%	$n=50$	84.9%	99.9%

Table 5. Experiments that Confirm the m-K Property and Run Out of Time Limit (n/a) with Variable n

	$m=1, K=5$		$m=2, K=5$	
	confirmed	n/a	confirmed	n/a
$n=10$	42.8%	0%	90.6%	0%
$n=20$	49.0%	0%	91.9%	0%
$n=30$	54.4%	6.5%	92.9%	1.2%
$n=50$	42.7%	41.9%	94.0%	6.0%

Tasks are assigned priorities according to the Rate Monotonic rule. If the lowest priority task τ_n in the taskset is schedulable, the taskset is abandoned; otherwise, we proceed with the weakly hard real-time analysis on τ_n. This configuration is designed to stress the weakly hard analysis, since even if the lowest priority task τ_n is schedulable there may exist other non-schedulable tasks with a smaller number of interfering higher priority tasks for the m-K analysis.

In the analysis of each taskset we defined a runtime limit of 1800 seconds: if the analysis takes more than 1800 seconds without terminating, we stop and report a failure of the m-K analysis.

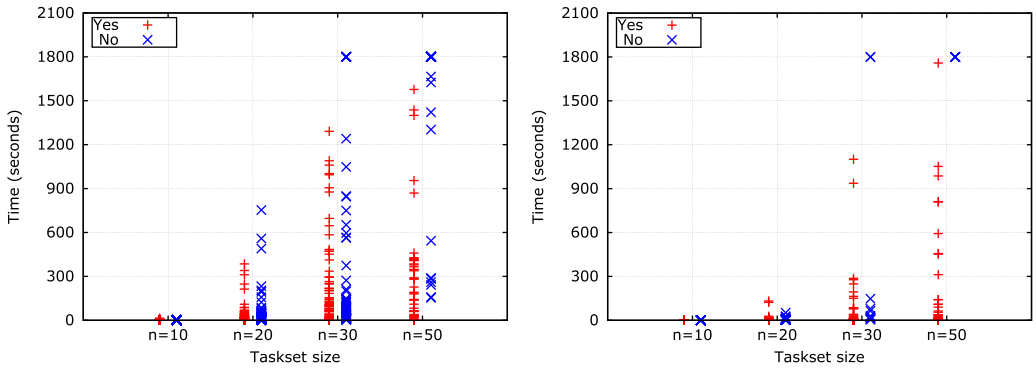
Deadline misses in a row. The m-K model discussed so far concerns the upper bound on the number of deadline misses (m) out of any K consecutive task activations. Another popular pattern for weakly hard schedulability analysis is to check if there are **more than m deadline misses in a row**, which is equivalent to analyze the m-K model with $K = m + 1$.

In the following, we evaluate the number of cases in which there are $K = 2$ and $K = 3$ consecutive deadline misses, when $U = 0.95$ and $n \in \{10, 20, 30, 50\}$. Results are shown in Table 4 and for every test case, the MILP solver returns its decision in less than a minute. Consecutive deadline misses seldom happen even when the total utilization is as high as 0.95. Another observation is that the fraction of cases with consecutive deadline misses is not sensitive with respect to the number of tasks in the set.

Varying the taskset size. Table 5 and Figure 3 show the experiment results with a variable taskset size, when $U = 0.85$. The weakly hard analysis confirms that a large portion of these non-schedulable tasks will never miss more than 1 deadline out of any 5 its consecutive activations. For example, when the taskset size is 20 or 30, the percentage of *1-5* feasible sets is around 50%. When m is increased to 2, more than 90% of the tested tasksets satisfy the specified m-K property in all cases.

On the other side, when the taskset size is very large ($n = 50$), for $m = 1$ a significant amount of tests exceed the runtime limit of 1800 seconds, which implies that a longer runtime is needed for such cases. Figure 3 depicts the time spent on the weakly hard analysis of each taskset: *Yes* labels that the corresponding m-K property is verified (*No* in the other case). The majority of analyses return the decision within 10 minutes.

Varying the problem window size. Table 6 contains the experiment results when varying the problem window size K, with $n = 10$ and $U = 0.85$. The problem window size K is a dominant



(a) $m = 1$ (b) $m = 2$

Fig. 3. Runtime results for $K = 5$.

Table 6. Percentage of Valid m-K Property with Variable K

	$m = 2$	$m = 3$		$m = 2$	$m = 3$
$K = 11$	54.4%	84.8%	$K = 14$	47.0%	73.6%
$K = 12$	51.3%	81.5%	$K = 15$	45.8%	66.4%
$K = 13$	49.2%	76.5%	$K = 20$	33.6%	56.1%

Table 7. Percentage of Valid m-K Property with Variable U

	$m = 2, K = 10$	$m = 3, K = 10$
$U = 0.80$	90.8%	99.1%
$U = 0.85$	60.8%	86.1%
$U = 0.90$	30.2%	50.6%
$U = 0.95$	6.8%	17.6%

factor with respect to the complexity of the analysis. Still, the results are promising and for more than one third of the tasksets, the number of deadline misses is bounded by at most $m = 2$.

Varying taskset utilization. Table 7 shows the percentage of tasksets that satisfy the m-K property with variable taskset utilization levels and a fixed taskset size $n = 10$.

Even when the taskset utilization is very high ($U = 0.90$), more than 30% of the non-schedulable tasks will not miss more than $m = 2$ deadlines within any sequence of $K = 10$ successive task activations. If we further increase m to 3, the tasksets satisfying the weakly hard property become half of the generated sets.

7 CONCLUSIONS

In this work, we propose a weakly hard schedulability analysis technique for fixed priority preemptive scheduling of a set of periodic tasks. Our approach applies to offset-free systems and is more general than previous works. The analysis is formulated as an MILP encoding, and its performance (regarding both the analysis precision and runtime efficiency) have been confirmed by extensive experiments.

A possible extension of the approach proposed in this paper could target multiprocessor systems with partitioned scheduling and shared resources as applicable to several automotive applications [1, 32].

REFERENCES

[1] Zaid Al-bayati, Youcheng Sun, Haibo Zeng, Marco Di Natale, Qi Zhu, and Brett Meyer. 2015. Task placement and selection of data consistency mechanisms for real-time multicore applications. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium*. 172–181.

[2] 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.

[3] 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.

[4] Karl-Erik Arzén, 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.

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

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

[7] Enrico Bini, Marco Di Natale, and Giorgio Buttazzo. 2008. Sensitivity analysis for fixed-priority real-time systems. *Real-Time Systems* 39, 1–3 (2008), 5–30.

[8] Alessandro Biondi, Marco Di Natale, Youcheng Sun, and Stefania Botta. 2016. *Moving from Single-core to Multicore: Initial Findings on a Fuel Injection Case Study*. Technical Report. SAE Technical Paper.

[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*. 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] Robert I. Davis, Ken W. Tindell, and Alan Burns. 1993. Scheduling slack time in fixed priority pre-emptive systems. In *Real-Time Systems Symposium. Proceedings*. IEEE, 222–231.

[14] José Luis Díaz, Daniel F. García, Kanghee Kim, Chang-Gun Lee, L. Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. 2002. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium. RTSS. 23rd IEEE*. IEEE, 289–300.

[15] Paul Emberson, Roger Stafford, and Robert I. Davis. 2010. Techniques for the synthesis of multiprocessor tasksets. In *Proceedings 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 6–11.

[16] 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.

[17] 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.

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

[19] Jung-Eun Kim, Tarek F. Abdelzaher, and Lui Sha. 2015. Budgeted generalized rate monotonic analysis for the partitioned, yet globally scheduled uniprocessor model. In *21st IEEE Real-Time and Embedded Technology and Applications Symposium, Seattle, WA, USA, April 13-16, 2015*. 221–231.

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

[22] 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.
[23] 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.
[24] M. M Hamdaoui and P. Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. In *IEEE Transactions on Computers*.
[25] 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.
[26] 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.
[27] 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.
[28] Lui Sha, Tarek Abdelzaher, Karl-Erik Arzén, 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.
[29] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. 1990. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers* 39, 9 (1990), 1175–1185.
[30] Damoon Soudbakhsh, Linh T. X. 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).
[31] Damoon Soudbakhsh, Linh T. X. 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.
[32] Alexander Wieder and Björn B. Brandenburg. 2013. Efficient partitioning of sporadic real-time tasks with shared resources and spin locks. In *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*. 49–58.
[33] 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.
[34] Yang Xu, Karl-Erik Arzén, Enrico Bini, and Anton Cervin. 2014. Response time driven design of control systems. *IFAC Proceedings Volumes* 47, 3 (2014), 6098–6104.
[35] Yang Xu, Karl-Erik Arzén, 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.

Received April 2017; revised June 2017; accepted June 2017