2018 IEEE Real-Time and Embedded Technology and Applications Symposium

Firmness Analysis of Real-Time Applications Under Static-Priority Preemptive Scheduling

Amir R. B. Behrouzian*, D. Goswami*, T. Basten*†, M. Geilen*, H. Alizadeh Ara*, and M. Hendriks†
*Eindhoven University of Technology, Eindhoven, The Netherlands
†ESI, Eindhoven, The Netherlands
Email: {a.r.baghban.behrouzian, d.goswami, a.a.basten, m.c.w.geilen, s.h.seyyed.alizadeh}@tue.nl, martijn.hendriks@tno.nl

***Abstract*—(m,k)-firm real-time tasks must meet the deadline of at least m jobs out of any k consecutive jobs to satisfy the firmness requirement. Scheduling of an (m,k)-firm task requires firmness analysis, whose results are used to provide system-level guarantees on the satisfaction of firmness conditions. We address firmness analysis of an (m,k)-firm task that is intended to be added to a set of asynchronous tasks scheduled under a Static-Priority Preemptive (SPP) policy. One of the main causes of deadline misses in periodic tasks running under an SPP policy is interference from higher priority tasks. Since the synchrony between the newly added task and higher priority tasks is unknown, the interference from the higher priority tasks is also unknown. We propose an analytic Firmness Analysis (FAn) method to obtain a synchrony that results in the maximum minimum number of deadline hit jobs in any k consecutive jobs of the task. Scalability of FAn is compared with that of existing work – a brute-force search approach – and a timed-automata model of the problem that is analysed using the reachability check of the UPPAAL model checker. Our method substantially reduces the complexity of the analysis.**

***Key words*: Deadline miss, (m,k)-firm real-time tasks, static-priority preemptive**

I. INTRODUCTION

Real-time systems are traditionally categorised into *hard*, *soft* and *firm* in terms of the nature of their deadlines. In contrast with hard real-time systems, in soft and firm real-time systems, an occasional Deadline Miss (DM) (i.e., finishing a job execution after its deadline) causes no catastrophic consequences [1]. There is no added value in executing a deadline missed job of firm real-time tasks in contrast with soft real-time tasks.

As a category of firm real-time tasks, (m,k)-firm tasks tolerate certain DMs with strict constraints on their distribution. That is, an (m,k)-firm task [2] must have at least m Deadline Hits (DHs) (i.e., a job meets its deadline) in any k consecutive jobs to satisfy system-level requirements. We consider real-time tasks governed by (m,k)-firmness conditions. We analyze if a job (of a task) is going to miss its deadline immediately after its release and for a DM, the job is assumed to be never executed. This scenario is particularly relevant for feedback control applications where executions after deadlines may lead to the use of outdated feedback signals and stability issues.

Schedulability analysis for shared resources is necessary to provide temporal predictability for real-time systems. Static-Priority Preemptive (SPP) scheduling is commonly used in many real-time software systems, such as AUTOSAR-based operating systems [3]. In an SPP system, fixed priorities are assigned to tasks and they can be preempted by activations of higher priority tasks. If preemptions cause *too much* delay in a task execution, the task misses its deadline. A firmness analysis aims to guarantee that the distribution of DHs is consistent with requirements of (m,k)-firm conditions.

One crucial factor in the number of DHs is synchrony among the tasks, which significantly influences the interference. In this regard, strictly periodic tasks are divided into two classes: synchronous and asynchronous [4]. They differ in assigning a value to the release time of the first job of the task. For synchronous tasks, the *first release time* is not fixed, relative to the other tasks, whereas for asynchronous tasks it is fixed[1]. With knowledge of the release time of the first job of asynchronous tasks, the interference from higher priority tasks becomes predictable.

In the process of mapping multiple hard and (m,k)-firm real-time tasks on a shared processor, the hard real-time tasks are usually mapped first in order to provide guarantees on their schedulability. Next, the (m,k)-firm tasks are mapped and they require further analysis to ensure satisfaction of their constraints. We consider a case in which a new (m,k)-firm asynchronous task is intended to be added to a set of asynchronous tasks running under an SPP policy. This is common in many mixed-criticality application scenarios where applications are incrementally mapped according to their criticality-level. We aim to identify the best-case release time of the newly added asynchronous task to obtain the *maximum minimum* (max-min) number of DHs in any window of k consecutive jobs. It is the lower bound for the maximum possible number of DHs and we refer to it as max-min DHs. While *maximizing* DHs is preferable for better performance

This paper has passed an Artifact Evaluation process. For additional details, please refer to http://2018.rtas.org/artifact-evaluation/. All results reported in this work can be reproduced using the material provided in https://github.com/behrouzian/Amir-Behrouzian.git.

[1]We follow the definition in [4]. If first release times of a set of tasks are not known, they may potentially be released simultaneously (synchronous tasks). For asynchronous tasks, first release times of a set of tasks are known and task releases (typically) do not happen simultaneously. Note that the task definitions are exactly opposite in the work reported in [5][6].

978-1-5386-5295-4/18/$31.00 ©2018 IEEE
DOI 10.1109/RTAS.2018.00037

of the system, a *minimum* number of DHs is required for a guaranteed lower bound on the number of DHs. This guarantee is obtained by considering the worst-case execution time for all tasks. We show that the number of DHs can only grow if execution time of one or multiple jobs (of the task under firmness analysis or the task with higher priorities) becomes less than its worst-case execution time.

Generally, processors run under a digital clock. The first release time of tasks can take any value that is an integer multiple of the clock cycle of the processor. Based on existing work [5] a brute-force approach on all these possibilities is required in order to obtain the max-min DHs in any k consecutive jobs of the newly added task. We show that there can be a prohibitively large number of possible first release times of the newly added task to allow for a brute-force search. We propose an analytic method, FAn, to obtain a first release time for the newly added task that leads to the max-min DHs in any k consecutive jobs of the task. FAn needs to consider only a small subset of all possibilities to compute the release time leading to the max-min DHs, making it inherently scalable compared to the brute-force approach.

The paper is organized as follows. Section II discusses related work. Section III presents background concepts. We first consider a case with only two synchronous tasks in Section IV, i.e., two tasks with unknown relative release time. It covers some key concepts that are used to analyse multiple asynchronous tasks with one new task to be added. For this simplified problem, we provide a temporal model of an SPP policy that allows us to predict whether a job will meet its deadline immediately after it is released. Next, we consider a window of k consecutive jobs of a task and obtain the max-min DHs in any k consecutive jobs. We extend this method to multiple asynchronous tasks in Section V. We obtain the first release time for a newly added (m, k)-firm task that leads to the max-min DHs in any k consecutive jobs.

For the evaluation in Section VI, we take multiple benchmark case studies in which we map an (m, k)-firm task under an SPP policy. We compare scalability of our method with that of an existing method in the literature [5] and a timed-automata model of the problem using UPPAAL [7]. Based on an exhaustive search, UPPAAL builds up a search space of all possible first release times of tasks and obtains the cases that cause max-min DHs. Finally, Section VII concludes.

II. Related Work

Hamdaoui [2] introduces the notion of (m, k)-firm deadlines in the context of scheduling messages in a network. Bernat [5] further extended the notion of (m, k)-firm deadlines to *weakly hard real-time tasks* by introducing different types of constraints on the distribution of DHs and DMs.

In the literature, (m, k)-firmness is studied from two different perspectives. One direction of work investigates the extraction of m and k parameters for a specific task based on the robustness of the systems [8][9][10]. Another line of work studies the satisfaction of given (m, k)-firmness conditions of a task in a schedule [2] [5] [11] [12] [13] [14] [15]. Our work falls into the second category. A best-effort scheduling algorithm is proposed in [2] for (m, k)-firm tasks under an SPP policy. The reported algorithm does not provide any guarantee on the number of DHs. Bernat [5] analyses the firmness of a task in a set of asynchronous tasks under an SPP policy with known first release times. The response time of all the jobs of the task in the first hyperperiod is obtained first. It then searches for the worst distribution of deadline miss patterns. The techniques proposed in [5] can be used for *all* possible first release times of the newly added task to obtain the max-min DHs. However, we show that this method can be prohibitively inefficient. We propose a method that reduces the size of the search space to a practical size. The method first reported in [5] is also used in recent literature in a different context, e.g. firmness analysis of asynchronous tasks migrating between different processors [11].

Kong [12] proposes a hierarchical dynamic priority-preemptive scheduling algorithm based on the history of the system in terms of the number of DMs before a decision point. Based on typical worst-case response time analysis on the execution trace of a system, Tobuschat [14] provides an upper bound on the number of DMs of a task running under an SPP policy. Behrouzian [15] proposes a method for firmness analysis of a periodic task running under a Time Division Multiple Access (TDMA) policy. For the given time slots allocated to the task in a TDMA policy, upper and lower bounds on the number of deadline misses are calculated. All this work assumes that the first release time of the task being analyzed is known. The reported analysis methods are not applicable when the goal is to find the optimal first release time of a task during design time, as in our work. We do not assume knowledge about the first release times of a task that is intended to be added to an existing SPP schedule.

III. Setup Under Consideration

A. Task Model

We consider a set $\Pi = \{\tau_i\}_{1 \leq i \leq N}$ of N periodic tasks. Each task $\tau_i = (C_i, T_i, D_i)$ has a worst-case execution time C_i, an activation period T_i and a relative execution deadline D_i. We assume that the deadline of a task is at most equal to its activation period, i.e. $D_i \leq T_i$. We further assume that no job is allowed to be executed after its deadline. We analyze whether a job of a task is going to miss its deadline immediately after its release and for a DM, the job is assumed to be never executed. This setup is particularly applicable for feedback control applications where executions after deadlines may lead to the use of outdated feedback signals, variable sensor-to-actuator delay and stability issues.

Every instance of a task that starts with a new activation is referred to as a *job*. The n^{th} job of the task τ_i is denoted by α_i^n. The activation time of each job is called the *release time* of the job. We model accessibility of a task τ_i to a resource at time t with the *Accessibility function* $l_i(t)$.

Definition 1 (Accessibility function)**.** *Accessibility function* $l_i : \mathbb{R}^+ \rightarrow \{0, 1\}$ *takes* 0 *if the resource is executing a task with a higher priority than* τ_i *at time* t *and* 1 *otherwise.*

In order to realize if a job released at t meets its deadline or not, we have to study the overall access of τ_i to the resource over the interval between t and $t + D_i$. This is the interval within which the job must be executed to meet its deadline. Next, we introduce a function that captures resource accessibility over this interval.

Definition 2 (Total accessibility function)**.** *Total accessibility function $g_i : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ specifies the total amount of time between t and $t + D_i$ that a job of τ_i released at time t may have access to the processor.*

$$g_i(t) = \int_t^{t+D_i} l_i(x)dx \tag{1}$$

where $l_i(t)$ is the accessibility function of τ_i.

In other words, $g_i(t)$ determines the resource service for τ_i in the next D_i time units. If τ_i releases a job at t, we can immediately decide whether that job will meet its deadline by comparing $g_i(t)$ and C_i. That is, if $g_i(t) \geq C_i$, the job that is released at t meets its deadline. Using this comparison over a period of time, we can obtain the intervals where tasks are prone to miss or meet their deadlines. The *Hit-zone function* models these intervals.

Definition 3 (Hit-zone function)**.** *Hit-zone function $h_i : \mathbb{R}^+ \rightarrow \{0, 1\}$ specifies the intervals in which a starting job of task τ_i will meet its deadline. We refer to these intervals as hit-zones.*

$$h_i(t) = \begin{cases} 0 & C_i > g_i(t) \\ 1 & C_i \leq g_i(t) \end{cases} \tag{2}$$

where C_i is the worst-case execution time of τ_i.

Definition 4 (Deadline Hit)**.** *A job of τ_i released at t is a Deadline Hit (DH) if $h_i(t) = 1$.*

If a job of τ_i is not a DH, it is referred to as a DM. For a given $h_i(t)$ of a task τ_i, a DM is recognized immediately after the task release. Therefore, it is used to skip any execution on a job that is a DM.

Definition 5 (Execution function)**.** *The execution function $E_i^n : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ specifies the overall time that α_i^n has been processed until time t.*

Note that, for a DH α_i^n that is released at time t, $E_i^n(t + D_i) = C_i$.

Definition 6 (Pending job)**.** *The job α_i^n released at t' is pending at time t if $E_i^n(t) < C_i$ where $t' \leq t < t' + D_i$.*

Definition 7 ((m, k)-firm task)**.** *The task τ_i is (m, k)-firm if the number of DHs in any window of k consecutive jobs is at least m.*

B. Synchronous and Asynchronous Tasks

Periodic tasks are divided into two main categories: synchronous and asynchronous. We follow the same definition as [4] for these two categories. For a synchronous task, the *first release* time is not fixed, relative to the other tasks in the set. That is, in a set of synchronous periodic tasks, relative release times of different tasks may get any value. On the other hand, the first release time is fixed for an asynchronous task. We define a new parameter O_i that denotes the first release time of an asynchronous task τ_i. In a set of asynchronous tasks under an SPP policy, O_i is relative to a reference time $t = 0$ that holds for all tasks in the schedule. Therefore, we define an asynchronous task as $\tau_i = (C_i, T_i, D_i, O_i)$.

C. SPP Scheduling Policy

We define an SPP scheduling policy for the task set $\Pi = \{\tau_i\}_{1 \leq i \leq N}$ as a set $SPP = \{(\tau_i, P_i)\}_{1 \leq i \leq N}$ of N pairs. Each pair (τ_i, P_i) assigns the priority P_i to the task τ_i. α_i^n released at t' is executed at any time t (where $t' \leq t < t' + D_i$) if $E_i^n(t) < C_i$ and there is no task with higher priority than τ_i pending.

IV. FAN FOR TWO SYNCHRONOUS TASKS

We first investigate whether synchronous tasks might have DMs in an SPP system or not. Traditionally, Worst Case Response Time (WCRT) analysis [16] is used to study schedulability of a set of synchronous tasks. Next, considering two synchronous tasks, a method is proposed to quantify the max-min number of DHs in any k consecutive jobs of the task with the lower priority.

A. Schedulability Analysis of Synchronous Tasks

Using WCRT analysis for synchronous tasks, regardless of the number of tasks, the worst possible release time of the tasks – also known as *critical instant* [4] – is considered for schedulability analysis of synchronous tasks. The critical instant for a synchronous task τ_i occurs when τ_i and all the tasks with higher priorities release a job at the same time. The basic idea behind WCRT analysis is that if the job of a task that is released at the critical instant meets its deadline, all other jobs of the task meet their deadlines.

Let us assume a critical instant for the task τ_i with a release of τ_i and all the tasks with higher priorities at $t = 0$. The WCRT of τ_i, which is denoted by $WCRT_i$, is the minimum $t > 0$ that satisfies the equation below [4].

$$t = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{t}{T_j} \right\rceil C_j, \tag{3}$$

where $hp(\tau_i)$ is the set of all the tasks with higher priorities than τ_i. Note that this solution considers no context switching overhead of the resource. A closed-form solution of Eq. 3 does not exist because of the nonlinear nature of the ceiling operator inside the summation. As an alternative, the fixed point iteration method [17] is used to solve Eq. 3 as follows.

$$\begin{cases} t^{(0)} = C_i \\ t^{(\gamma+1)} = C_i + \sum_{\tau_j \in hp(\tau_i)} \left\lceil \frac{t^{(\gamma)}}{T_j} \right\rceil C_j \end{cases} \tag{4}$$

where $\gamma \in \{0, 1, 2, ...\}$. These iterations have to be continued by incrementally increasing the value of γ until the value of $t^{(\gamma)}$ does not change anymore, i.e. $WCRT_i = t^{(\gamma+1)} = t^{(\gamma)}$. That is, the last result is the actual $WCRT_i$. Note that Eq. 3

converges only if the overall resource utilization is at most 1 [6].

If $WCRT_i > D_i$, τ_i is therefore not schedulable in the SPP system. For the task τ_i that is not schedulable in an SPP system, release at the critical instant is a sufficient but not a necessary condition for the release of τ_i to lead to a DM. In the next subsection, we discuss the necessary conditions for the release time of τ_i relative to those of higher priority tasks such that τ_i meets or misses its deadline.

B. Hit-Zone Calculation

Let us consider two synchronous periodic tasks $\tau_1 = (C_1, T_1, D_1)$ and $\tau_2 = (C_2, T_2, D_2)$ under an SPP policy with priorities $P_2 > P_1$, where τ_1 is not schedulable, i.e. $WCRT_1 > D_1$ (see Section IV-A). This is not the only scenario that leads to a DM for τ_1. By nature, in an SPP system, different relative release times of τ_1 may lead to different response times, which may or may not be shorter than, equal to or longer than the task deadline D_1.

From Section III, accessibility function $l_1(t)$ is obtained by information about the time intervals in which the resource is executing the tasks with higher priority, i.e. τ_2. Since we consider two synchronous tasks, without loss of generally, we can assume that the first release of τ_2 occurs at $t = 0$. From the assumption that τ_2 is periodic with the period of T_2, we conclude that $l_1(t)$ is also periodic with the same period. $l_1(t)$ for one period is obtained as follows.

$$l_1(t) = \begin{cases} 0 & 0 \leq t < C_2 \\ 1 & C_2 \leq t < T_2 \end{cases} \tag{5}$$

The total accessibility function $g_1(t)$ is then obtained from Eq. 1 by substituting D_i with D_1 and $l_i(t)$ with $l_1(t)$. The hit-zone function h_1 then specifies the necessary condition for meeting the deadline of a job of τ_1 that is released at time t by comparing C_1 with $g_1(t)$ as in Eq. 2.

Example IV.1. *Let us consider an example of two synchronous periodic tasks $\tau_1 = (9, 17, 16)(\times 100\mu s)$ and $\tau_2 = (6, 13, 11)(\times 100\mu s)$ in an SPP policy with $P_2 > P_1$.*

Fig. 1(a) illustrates accessibility function $l_1(t)$ against t for the task set that is obtained using Eq. 5. Fig. 1(b) illustrates the total accessibility function $g_1(t)$ over t for Ex. IV.1. The dashed straight line on Fig. 1(b) shows the worst-case execution time of τ_1, C_1. Fig. 1(c) illustrates the hit-zone function $h_1(t)$ against t for the total accessibility function $g_1(t)$.

So far, we have formulated the resource service for a single job of a task. Next, we investigate the resource availability for a window of k consecutive jobs.

C. Quantification of the Max-Min DHs

With a given hit-zone function $h_1(t)$, we can decide whether a job of τ_1 that is released at t will meet its deadline or not, i.e. the job will meet the deadline only if $h_1(t) = 1$. However, we are interested in multiple releases of τ_1, particularly a window of k consecutive jobs of τ_1. We define the *firmness function* as a function that captures the number of DHs in a window of k consecutive jobs based on the *offset* of the window. The offset is the release time of the first task in the window relative to time $t = 0$ in a given hit-zone function $h_1(t)$.

Definition 8 (Firmness function)**.** *Given a hit-zone function $h_i(t)$, the firmness function $m_i : \mathbb{R}^+ \times \mathbb{N} \to \mathbb{N}$ denotes the number of DHs in a window of k consecutive jobs of τ_i.*

$$m_i(O, k) = \sum_{m=0}^{k-1} h_i(O + mT_i) \tag{6}$$

O is the offset of the window.

Fig. 1(d) shows the firmness function $m_1(O, k)$ against the offset O for Exp. IV.1 with $k = 10$. This graph is obtained by discretizing the offset O over one period of $g_1(t)$ shown in Fig. 1(b). The value of m for a given O indicates the number of jobs in a window of k consecutive jobs that are released inside the hit-zone if the first job of the window is released at O. We discuss about the points that are marked by stars in Fig. 1(d) later in Section IV.

Based on Theorem 1, considering the worst-case execution time for both tasks, results in the lower bound for the number of DHs for any given first release time of τ_1. That justifies the *min* part of max-min DHs that we aim to obtain.

Theorem 1. *Consider two applications $\tau_1 = (C_1, T_1, D_1)$ and $\tau_2 = (C_2, T_2, D_2)$ running under SPP policy with priorities $P_2 > P_1$. When the execution times C_1' and C_2' of the two tasks τ_1 and τ_2 are smaller than their worst-case execution times, i.e. $C_1' \leq C_1$ and $C_2' \leq C_2$, results in a hit-zone function $h_1'(t)$ where $h_1'(t) \geq h_1(t)$ for any $t \geq 0$.*

Proof. Let us first assume that only the execution time of the task with higher priority is smaller than its worst case execution time, i.e., $C_1' = C_1$ and ${C_2}' < C_2$. The corresponding value of accessibility function, total accessible function and hit-zone function of τ_1 considering C_2' is denoted by $l_1'(t)$, $g_1'(t)$ and $h_1'(t)$ and considering C_2 is denoted by $l_1(t)$, $g_1(t)$ and $h_1(t)$. From Eq. 5 and Eq. 1

$$\forall t \geq 0, l_1'(t) \geq l_1(t), g_1'(t) \geq g_1(t).$$

Therefore,

$$\forall C_2' < C_2, h_1'(t) \geq h_1(t).$$

Next, we assume that $C_1' < C_1$ and ${C_2}' \leq C_2$. From Eq. 2, we obtain $h_1''(t) \geq h_1(t)$ where $h_1''(t)$ is the corresponding hit zone function for C_1'. □

Theorem 1 implies that the number of DHs for any first release time of O_1, might increase with decreasing the execution time of at least one of τ_1 or τ_2 from their worst-case execution time.

The max-min number of DHs can be obtained by computing the number of DHs for all possible O considering worst-case execution time for both tasks. Using the following theorem we can reduce the search space to a limited interval.

Theorem 2. *Firmness function $m_1(O, k)$ is periodic with the period of T_2.*

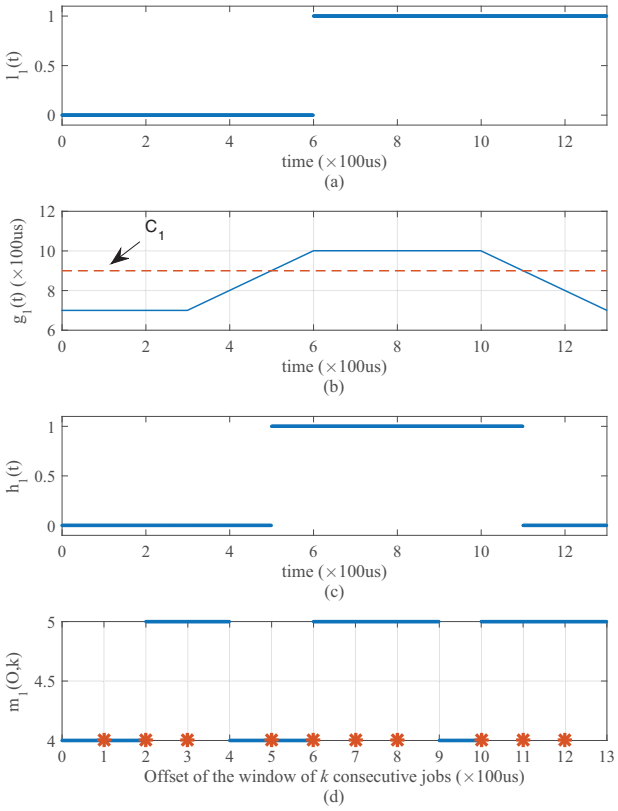

Fig. 1: Firmness analysis of τ_1 in Exp. IV.1. (a) accessibility function $l_1(t)$. (b) total accessibility function $g_1(t)$. The dashed line shows the worst-case execution time of τ_1. (c) hit-zone function $h_1(t)$. t_{bng} and t_{end} denote the beginning and the end of a hit zone in the first period of $h_1(t)$.(d) firmness function $m_1(O, 10)$. O_{cnd} is shown with stars.

Proof. Since $l_1(t)$ is periodic with the period of T_2, the substitution of $x' = x - T_2$ in Eq. 1 yields

$$g_1(t+T_2) = \int_{t+T_2}^{t+T_2+D_1} l_1(x)dx = \int_{t}^{t+D_1} l_1(x'+T_2)dx' = \int_{t}^{t+D_1} l_1(x')dx' = g_1(t).$$

Therefore, from Eq. 2, $h(t+T_2) = h(t)$ and Eq. 6

$$m_1(O+T_2,k) = \sum_{m=0}^{k-1} h_1(O+T_2+mT_1) = \sum_{m=0}^{k-1} h_1(O+mT_1) = m_1(O,k).$$

□

Therefore, it is sufficient to consider offset O of τ_1 to be between $t = 0$ and $t = T_2$ in order to cover all possible patterns of DHs in k consecutive jobs of τ_1. In view of the digital clock, this further implies that O may take as many values as the length of T_2 in clock cycles. As already stated, this can be prohibitively large. Next, we further reduce the size of the search space.

Using the Finite Point (FP) method [15], we first find finitely many candidates for the offset of the window of k consecutive jobs of which at least one leads to the maximum number of releases inside the hit-zone. The philosophy behind the FP method is that while increasing the offset O from 0 to T_2 the value of $m_1(O,k)$ increases when the release time of at least one of the jobs inside the window enters a hit-zone. The following formal statement is derived from the FP method [15],

$$\forall t \in [0, T_2), \exists n \in \mathbb{N}:$$
$$m_1(t+\epsilon, k) > m_1(t,k) \Rightarrow h_1(nT_2+t+\epsilon) > h_1(nT_2+t)$$

for every $\epsilon > 0$ [2] and $n \in \{0, 1, 2, ...\}$. In one of these increases of $m_1(O,k)$, it must reach the maximum possible value. Therefore, we find the set O_{cnd} of offsets o_{cnd} for a window of k consecutive tasks in which one of the releases of τ_1 in the window is at the beginning of a hit-zone where

$$o_{cnd} = \begin{cases} t_{bgn} - mod & t_{bgn} \geq mod \\ T_2 + t_{bgn} - mod & t_{bgn} < mod \end{cases} \quad (7)$$

where

$$mod = nT_1 - T_2 \left\lfloor \frac{nT_1}{T_2} \right\rfloor \quad (8)$$

and $\{n \in \mathbb{Z} : 0 \leq n < k-1\}$. Let T_{bgn} be the set of the starting times t_{bgn} of the hit-zone $h_1(t)$ in its first period. We obtain T_{bgn} by solving the following equation

$$T_{bgn} = \{t_{bgn} \in [0, T_2) : \frac{dg_1}{dt}|_{t_{bgn}} > 0 \wedge h_1(t_{bgn}) = 1 \wedge h_1(t_{bgn} - \epsilon) = 0\} \quad (9)$$

where $\frac{dg_1}{dt}|_{t_{bgn}}$ denotes the derivative of $g_1(t)$ at $t = t_{bgn}$.

All t_{bgn} for Example IV.1 is shown in Fig. 1(c). The set O_{cnd} for Ex. IV.1 is shown with stars in Fig. 1(d). As it is shown in the figure, any increase in the number of DHs, i.e. the value of $m_1(O, 10)$, occurs when there is a star. This further means that at least one of the 10 consecutive jobs is at the beginning of a hit-zone.

Finally, we have to determine which offset in the set O_{cnd} leads to the maximum number of DHs. This can intuitively be obtained by comparing all the values of $m_1(o_{cnd}, k)$ for all $o_{cnd} \in O_{cnd}$. In other words,

$$m_{1-max}(k) = \max_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k) \quad (10)$$

[2]While the FAn method is continuous in time, we consider discrete time with the smallest time step ϵ equal to one clock cycle.

Algorithm 1 Firmness analysis of two synchronous tasks

Input $\Pi = \{\tau_1, \tau_2\}$, $SPP = \{spp_1, spp_2\}$, $spp_1 = (\tau_1, P_1)$, $spp_2 = (\tau_2, P_2)$, $P_2 > P_1$, $\tau_1 = (C_1, T_1, D_1)$, $\tau_2 = (C_2, T_2, D_2)$, k

Output $m_{1-max}(k)$ and o_{max}

Phase 1 – schedulability analysis

1: Calculate $WCRT_1$ ▷ Eq. 3
2: **if** $WCRT_1 \leq D_1$ **then** ▷ if the task is schedulable
3: $m_{1-max}(k) = k$
4: **else**

Phase 2 – obtain hit-zone function

5: Calculate the accessibility function $l_1(t)$ ▷ Eq. 5
6: Calculate the total accessibility function $g_1(t)$ ▷ Eq. 1
7: Calculate the hit-zone function $h_1(t)$ ▷ Eq. 2

Phase 3 – compute $m_{1-max}(k)$

8: $mod = nT_1 - T_2 \lfloor nT_1 \,/\, T_2 \rfloor$
9: Obtain starting times of hit-zone T_{bgn} in $[0, T_2)$ ▷ Eq. 9
10: Obtain candidate offsets O_{cnd} ▷ Eq. 7
11: $m_{1-max}(k) = \max_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$
12: $o_{max} = \operatorname{argmax}_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$
13: **end if**

where $m_{1-max} : \mathbb{R}^+ \times \mathbb{N} \to \mathbb{N}$ is the maximum possible number of DHs in k consecutive jobs of τ_1 . Let $o_{max} \in O_{cnd}$ be the offset that results in m_{1-max} DHs, i.e., $o_{max} = \operatorname{argmax}_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$. Alg. 1 summarizes the above firmness analysis for two synchronous tasks.

V. FAn for Multiple Asynchronous Tasks

In this section, we quantify the max-min number of DHs for an asynchronous task $\tau_1 = (C_1, T_1, D_1, O_1)$ in a set of multiple asynchronous tasks. We assume that τ_1 is intended to be added to a set of asynchronous tasks that are already scheduled under an SPP policy. In contrast to other tasks that are already scheduled, the first release time of τ_1 has not been decided.

A. Extension of FAn for Multiple Asynchronous Tasks

The fact that the release times of all jobs of tasks with higher priorities than τ_1 are known, allows us to compute the relative times when the resource is not accessible to τ_1, regardless of the first release time of τ_1. This triggers the idea of considering the combination of all the tasks with higher priorities than τ_1 as one task (from τ_1 perspective) with a higher priority than τ_1 and a period equal to the *hyperperiod of level* 1^+, H_{1^+}.

Definition 9. *(Hyperperiod of level i^+) For a given set of asynchronous tasks, the hyperperiod of level i^+, H_{i^+}, is the hyperperiod of activation periods of all tasks with priorities higher than τ_i. Formally,*

$$H_{i^+} = lcm\{T_j | \tau_j \in hp(\tau_i)\}$$

where $hp(\tau_i)$ is the set of all tasks with priorities higher than τ_i.

The existence of the hyperperiod H_{i+} is justifiable since the activation periods T_i of the tasks are integer multiples of the system clock cycle. Therefore, we can assume that we have two synchronous tasks: the combination of all tasks with higher priorities than τ_1 as one task and τ_1 as another task. This inspires us to use the method that is proposed in Section IV. However, there is a substantial difference between the two cases. The task that is the combination of all tasks with higher priorities than τ_1 executes several times (as individual tasks are executed at different points of time) in its period H_{1^+}. This causes two major changes in the flow of Alg. 1. First, since the execution of the tasks with higher priorities does not happen at once, we cannot define any critical instant as we did for synchronous tasks. Therefore, we cannot initially run a fast schedulability analysis to realize whether there might be some deadline misses for the task with the lowest priority, as it is the case for synchronous tasks [4]. Second, the calculation of the accessibility function $l_1(t)$ is not as easy as what is presented in Section IV. This is explained later in Section V-B.

Example V.1. *Let us consider an example in which task $\tau_1 = (1.5, 13, 11, O_1)(\times 100\mu s)$ is intended to be added to a set of tasks $\Pi = \{\tau_2, \tau_3\}$ where $\tau_2 = (3, 8, 7, 4)(\times 100\mu s)$ and $\tau_3 = (2, 6, 3, 0)(\times 100\mu s)$ running under an SPP policy with priorities $P_3 > P_2 > P_1$. τ_1 has firmness condition $(8, 10)$. Due to the fact that the relative first release time between τ_2 and τ_3 is given we assume the combination of these tasks as a high priority task for τ_1. Fig. 2(a) shows the execution of τ_2 and τ_3 in the first hyperperiod of level 1^+, $H_{1^+} = lcm\{6, 8\} = 24$. As in Section IV, without loss of generality, we assume that the first release time of the task with the highest priority i.e. τ_3 is the time reference $t = 0$. Some features that are shown in Fig. 2, e.g. $idle_3$ and so on, are explained later in Section V-B.*

B. Hit-Zone Calculation

Let us assume that τ_1 is the task that is intended to be added to a set of asynchronous tasks. Accessibility function $l_1(t)$ for τ_1 can be obtained as follows

$$l_1(t) = \begin{cases} 0 & {B_j}^n \leq t < {R_j}^n \\ 1 & otherwise \end{cases} \tag{11}$$

where ${B_j}^n$ and ${R_j}^n$ are the release time and response time of ${\alpha_j}^n$, i.e. the n^{th} job of τ_j. Here $\tau_j \in hp(\tau_1)$ and $n = \{1, 2, ..., N_{j-hyp}\}$ where $N_{j-hyp} = H_{1^+} \,/\, T_j$ is the number of releases of τ_j within the first hyperperiod of level 1^+, H_{1^+}. The release time of each job can intuitively be obtained by ${B_j}^n = nT_j + O_j$. ${R_j}^n$ equals to the minimum value of $t > 0$ that satisfies the following equation [5]:

$$t = nC_j + Intf_j(t) + Idle_j({B_j}^n). \tag{12}$$

This equation consists of three terms: nC_j is the execution time of the first n jobs of τ_j. The interference function

$Intf_j(t)$ is the overall time that the resource has been executing releases of tasks with higher priorities than τ_j before t. $Idle_j({B_j}^n)$ is the overall time when there was no demanded workload from τ_j and tasks with higher priorities than τ_j before ${B_j}^n$. $Idle_j({B_j}^n)$ may be used to execute the tasks with lower priorities than τ_j, e.g. τ_1.

Fig. 2(a) shows the busy window for tasks τ_2 and τ_3 in Ex. V.1. The intervals whose summation results in $Intf_j(t)$ and $Idle_j({B_j}^n)$ is shown as In_j and Id_j respectively. Note that since τ_3 is the highest priority task, there is no time interval to be considered for In_3. In Eq. 12, the interference function $Intf_j(t)$ is substituted as follows.

$$Intf_j(t) = \sum_{\tau_{jj} \in hp(\tau_j)} \left\lceil \frac{t - O_{jj}}{T_{jj}} \right\rceil C_{jj}, \tag{13}$$

where $hp(\tau_j)$ is the set of tasks with priorities higher than τ_j, i.e. $P_1 < P_j < P_{jj}$. Eq. 13 can be solved using the fixed point iteration method.

Next, we compute $Idle_j({B_j}^n)$. Bernat [5] proposes a method to obtain $Idle_j({B_j}^n)$ by computing maximum execution time $\bar{C}$ for a virtual task $\bar{\tau} = (\bar{C}, \bar{T} = t, \bar{D} = t, \bar{O} = 0)$– with lower priority than τ_j–such that $\bar{\tau}$ is schedulable i.e. $t \leq \bar{D}$. However, the complication of obtaining the range of values of $\bar{C}$ to be checked from 0 to t [6] makes this method relatively inefficient for our case. From the fact that in our method the response times of all jobs of all tasks with higher priorities than τ_1 have to be obtained in the first hyperperiod H_{1^+} in order to obtain $l_1(t)$ (see Eq. 11), we propose an algorithm for calculation of $l_1(t)$ in which $Idle_j({B_j}^n)$ is calculated faster than that of [5]. This algorithm is shown in Alg. 2-Phase 1 and explained below.

We take an iterative process to obtain $l_1(t)$. That is, we consider a function $l'_1(t)$ that initially has the value of $l'_1(t) = 1$ and is updated in several steps to reach $l'_1(t) = l_1(t)$. We start with the first job of the highest priority task (for which $Idle_j({B_j}^n) = 0$ because it is released at $t = 0$) as shown in Alg. 2-Phase 1 lines 3 and 5. In each step (i.e. inner for loop of Alg. 2-Phase 1) we update the value of $l'_1(t)$. That is, in each loop we obtain ${B_j}^n$ and ${R_j}^n$ and apply $l'_1(t) = 0$ for ${B_j}^n \leq t < R_j^n$. Then $Idle_j(B_j^{n+1})$ can be obtained as follows.

$$Idle_j(B_j^{n+1}) = \int_0^{B_j^{n+1}} l'_1(t)dt \tag{14}$$

That is, starting from the first job of the highest priority task (for which we know $B_j^1 = 0$ and $R_j^1 = C_j$), we first update $l'_1(t)$ from Eq. 11. Next, we obtain idle time of the next job i.e. $Idle_j(B_j^{n+1})$. Finally, we obtain the response time of the next job, i.e. R_j^{n+1}, using Eq. 12 and Eq. 13. This process continues until the last job of the second lowest priority task – which has one priority higher than that of τ_1 – in one hyperperiod H_{1^+} (see Alg. 2-Phase 1-line$\{7, 12\}$). After calculation of the response time of the second lowest priority task we update $l'_1(t)$ for the last time and as the result $l'_1(t) = l_1(t)$. Note that in Eq. 12, since $Idle_j(B_j^n)$ is independent of t, it is computed only one time for each release of τ_j. Fig. 2(b) shows $l_1(t)$ for τ_1 in Ex. V.1. Note that $Idle$ intervals of the lowest priority task in the set of tasks with higher priorities than τ_1, i.e. Id_2 in Ex. V.1, are the intervals that τ_1 can have access to the resource, i.e. $l_1(t)$.

Once the accessibility $l_1(t)$ is calculated, total accessibility function $g_1(t)$, and hit-zone function $h_1(t)$ can be obtained from Eq. 1 and Eq. 2, respectively. Fig. 2(c) and Fig. 2(d) shows $g_1(t)$ and $h_1(t)$ for Ex. V.1.

C. Computation of the Max-Min DHs

As for the case with two synchronous tasks, we next obtain the set $T_{bgn} = \{t_{bgn}\}$ of all the starting times of hit-zone $h_1(t)$ in the first hyperperiod H_{1^+}. t_{bgn} is formally obtained from Eq. 9 by substituting T_j with H_{1^+} and $h_i(t)$ with $h_1(t)$. Based on the FP method (see Section IV-C), at least one of the offsets $o_{cnd} \in O_{cnd}$ of a window of k consecutive jobs of τ_1 leads to the max-min number of DHs. o_{cnd} can be obtained from Eq. 7 where mod is computed as follows.

$$mod = nT_1 - H_{1^+} \left\lfloor \frac{nT_1}{H_{1^+}} \right\rfloor \tag{15}$$

and $\{n \in \mathbb{Z} : 0 \leq n < k - 1\}$. Fig. 2(e) depicts $m_1(O, 10)$ for τ_1 in Ex. V.1 that is obtained by discretizing time with the step of one clock cycle. All o_{cnd} for τ_1 in Ex. V.1 are shown with stars in the figure. As shown in the figure, all increases in the value of $m_1(O, 10)$ occur when there is at least one star located. Substituting o_{cnd} in Eq. 6 results in the number of DHs in the window of k consecutive jobs of τ_1 i.e., $m_1(o_{cnd}, k)$. The maximum value of $m_1(o_{cnd}, k)$ is the max-min DHs i.e., $m_{1-max}(k)$. Formally, $m_{1-max}(k) = \max_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$. $o_{max} \in O_{cnd}$ is the offset that results in max-min DHs, i.e., $o_{max} = \text{argmax}_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$. Alg. 2 summarizes the FAn method for firmness analysis of an (m, k)-firm task τ_1 in an SPP system with multiple asynchronous tasks with higher priorities than that of τ_1 as explained in this section.

VI. Experimental Results

We compare the scalability of the FAn method that is proposed in this work with that of two existing approaches: (i) A Brute-Force (BF) search approach on all possible patterns of missed and met deadlines in k consecutive jobs which is inspired by [5] and (ii) a reachability analysis based on a Timed-Automata (TA) model of the system. The first approach yields exact results whereas the second approach – due to the technical limitations of the method (see Section VI-B) – theoretically yields conservative results which may be different than that of the exact solution. The FAn method scales better than both of the brute-force search and timed automata approaches as discussed later in this section.

We first explain the BF and TA methods. Next, we introduce the benchmark case studies that are considered for our experimental results. Moreover, we propose hypothesis to obtain the complexity of all three methods, i.e. FAn, BF and TA. Finally, we illustrate the scalability of all the methods with three of

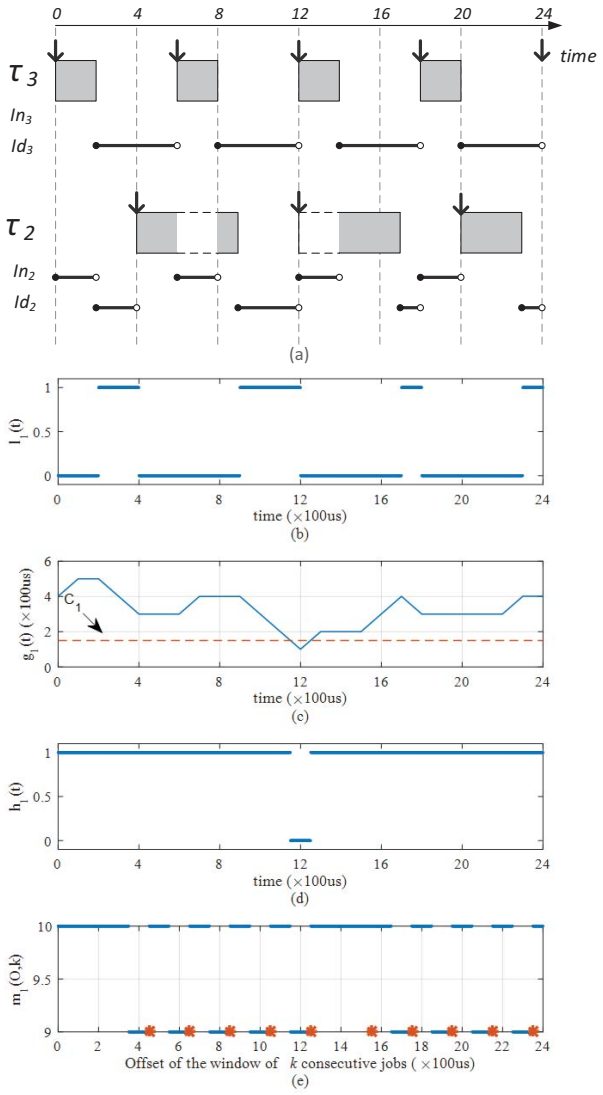

Fig. 2: Firmness analysis of τ_1 in Ex. V.1. (a) execution of τ_2 and τ_3 in the first hyperperiod of level 1^+. (b) accessibility function $l_1(t)$. (c) total accessibility function $g_1(t)$. The dashed line shows the worst-case execution time of τ_1. (d) hit-zone function $h_1(t)$. (e) firmness function $m_1(O,k)$. O_{cnd} is shown with stars.

the experimental setups that we considered for the evaluation of the complexity hypothesis.

A. Brute-Force Search Approach

Bernat [5] analyses the firmness of a (m,k)-firm task, τ_1, for a given first release time. In such a case, the interference from higher priority tasks on any job of τ_1 is known. Using this method for the firmness analysis of an (m,k)-firm task with an unknown first release time, the challenge is to ensure that all the possible interferences are considered before providing any guarantee on the max-min DHs. We use the following setup that is inspired from [5].

Algorithm 2 Firmness analysis of asynchronous tasks

Input $\Pi = \{\tau_1, \tau_j\}$, $SPP = \{spp_1, spp_j\}$, $spp_1 = (\tau_1, P_1)$, $spp_j = (\tau_j, P_j)$, $P_j > P_1$, $\tau_1 = (C_1, T_1, D_1, O_1)$, $\tau_j = (C_j, T_j, D_j, O_j)$, k
Output $m_{1-max}(k)$ and o_{max}

Phase 1 – hit-zone formulation

$H_{1^+} =$ LCM of P_j ▷ LCM: Least Common Multiple
$\forall 0 \leq t < H_{1^+}, l'_1(t) = 1$
for all $\tau_j \in hp(\tau_1)$ **do**
  $N_{j-hyp} = H_{1^+} \,/\, T_j$ ▷ N_{j-hyp}: # of jobs in a H_{1^+}
  for $n = 1 : N_{j-hyp}$ **do**
    $B_j^n = (n-1)T_j + O_j$
    $R_j^n = \sum_{\tau_{jj}} \lceil (R_j^n - O_{jj}) \,/\, T_{jj} \rceil C_{jj} + nC_j + Idle$
    $l'_1(t) = 0$ for $B_j^n \leq t < R_j^n$
    $Idle = \int_0^{B_j^{(n+1)}} l'_1(t)dt$
  end for
end for
$l_1(t) = l'_1(t)$
Calculate the total accessibility function $g_1(t)$ ▷ Eq. 1
Calculate the hit-zone function $h_1(t)$ ▷ Eq. 2

Phase 2 – compute $m_{1-max}(k)$

$mod = nT_1 - H_{1^+} \lfloor nT_1 \,/\, H_{1^+} \rfloor$
Obtain the starting time of hit-zone T_{bgn} between 0 and H_{1^+} ▷ Eq. 9 substituting $T_j \leftarrow H_{1^+}$ and $h_j \leftarrow h_1$
Obtain the candidate offsets for maximum DHs ▷ Eq. 7
$m_{1-max}(k) = \max_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$
$o_{max} = \operatorname{argmax}_{o_{cnd} \in O_{cnd}} m_1(o_{cnd}, k)$

Given the fact that the hit-zone function $h_1(t)$ is periodic with the period of H_{1^+}, all possible combinations of k consecutive jobs of τ_1 are obtained by considering all possible values of $0 \leq O_1 < H_{1^+}$ (see Section V). Moreover, from the fact that any execution commencement in a shared processor happens at the discrete points aligned with the starting of clock periods, we have a finite number of possible O_1 in the first hyperperiod H_{1^+}. Therefore, considering all possible first release times of τ_1 from 0 to H_{1^+}, we obtain all possible patterns of DHs and DMs in k consecutive jobs of τ_1. Then we obtain the max-min DHs by comparing all the results. Note that, we consider the worst-case execution time for all the tasks in order to obtain *min* DHs for any given first release time of τ_1.

B. Timed-Automata Model

The nature of an SPP system allows us to model the system using timed automata [18]. The complexity of a timed-automata verification using UPPAAL [7] highly depends on the complexity of the model itself. Therefore, it is unfair to compare the run-time of the TA method with that of BF and FAn methods. However, we compare the complexity of our timed-automata model – related to the parameters that we believe exist in a typical timed-automata model of an SPP system – with that of the BF and FAn method. [19] uses a

TABLE I: CCCA system parameters (time in ms)

Task	$\tau_i = (C_i, T_i, D_i, O_i)$	P_i	m	k
Breaking control (hard)	$(5, 30, 28, 0)$	4	-	-
Collision avoidance (hard)	$(15, 50, 48, 19)$	3	-	-
Engine control ((m,k)-firm)	$(15, 30, 28, 12)$	2	140	170
Display control ((m,k)-firm)	$(25, 50, 48, 7)$	1	140	170

reachability test on a similar timed-automata model of an SPP system to address the schedulability problem, whereas we use timed automata for firmness analysis of an (m, k)-firm task. The reachability check of the UPPAAL [7] model checker is used to analyse the timed-automata model.

We create a timed-automata model of a set of asynchronous tasks running under an SPP policy, including the (m, k)-firm task that is under firmness analysis. Each task is modeled with two automata. One automaton keeps track of release times of the task and another automaton counts the overall time that a job has been executed. Moreover, one automaton decides which task will be executed next – every time that a task releases a job. This model allows us to consider non-deterministic choices with respect to discrete events of the system and the progress of time. Stopwatch automata [20] which are a generalization of timed automata are used to model preemption by stopping the clock that represents the overall time that a job has been executed. Analysis of stopwatch automata is based on an over-approximation of the state space for an analysis. That is, the result of this model is not exact in general. A comparison between UPPAAL results and those of the FAn method shows no over-approximation in the final results of our experiments though. The only non-deterministic choice in the timed-automata model is the first release time of the task under firmness analysis.

C. Experimental Setup

For our experimental setup, we took multiple sets of asynchronous tasks and obtained the max-min DHs for (m, k)-firm tasks. The parameters for our experiments are inspired by the Cruise Control with Collision Avoidance (CCCA) system [11]. The CCCA system consists of Breaking Control (BC), Engine Control (EC), Collision Avoidance (CA) and Display Control (DC). Table I shows the parameters for each task aligned with our task definition in Section III. BC and CA have hard deadlines whereas EC and DC have (m, k)-firm constraints. Note that, despite the fact that the first release time of (m, k)-firm tasks are specified, all possible values for this parameter are considered in a firmness analysis. We adapted some parameters in some combinations of the tasks to have a setup with features that are required for our experiments. For example, we adapted periods of tasks to obtain a wide range for the hyperperiod of the tasks. Moreover, we took multiple instances of one task with different parameters when we required to have a combination of more than four tasks. In each experiment we analyse the firmness of one of the (m, k)-firm tasks, i.e. EC and DC.

We implemented the FAn and the brute-force method in MATLAB. A system equipped with a $2.6GHz$ quad core CPU and 4GB RAM is used for experiments.

TABLE II: Complexity of firmness analysis methods

Method	Complexity	Proper use case
FAn	$O(N_{hp(\tau_1)} H_{1^+})$	high # of tasks + long hyperperiod
BF	$O(k H^2_{1^+} N_{hp(\tau_1)} f)$	short hyperperiod
TA	$O(k^2 H^3_{1^+} N^c_{hp(\tau_1)})$	-

D. Scalability Analysis

Table II compares the complexity of the three methods that are considered in this section. It further specifies the case studies in which a particular method is preferred in terms of low run-time.

The main computationally demanding process in the FAn method is the calculation of the hit-zone function. As it is seen in Alg. 2, this process consists of two *nested for-loops* which scales linearly with the number of tasks with higher priorities than τ_1, i.e. $N_{hp(\tau_1)}$, and hyperperiod H_{1^+} of them. Therefore, the FAn method has a complexity of $O(N_{hp(\tau_1)} H_{1^+})$.

The same process of calculating the hit-zone function exists for BF method. Moreover, the run-time of the BF method depends on the number of clock cycles in a hyperperiod, i.e. $O(H_{1^+} f)$ where f is the clock frequency of the underlying operating system. Besides, for any given first release time of τ_1, the release time of all k consecutive jobs and the value of hit-zone function $h_1(t)$ is calculated. Therefore, this approach has a complexity of $O(k H^2_{1^+} N_{hp(\tau_1)} f)$.

The run-time of a timed-automata model depends on the size of the state space made during reachability analysis. Our experiments show that the range of the non-deterministic choice, i.e. H_{1^+} in our case, has a polynomial effect on the run-time of the method. Moreover, we believe that any timed-automata model of the system would consider the release of a job as a new state. Therefore, the state space scales with the number of jobs in a hyperperiod which depends on $N_{hp(\tau_1)}$ and H_{1^+}. Besides, reachability analysis searches for a path in the state space that increase the number of DHs with an upper bound of k and it terminates building up the state space once k jobs have been released. Our results confirm that the run-time of the timed-automata method increases quadratically with k and polynomially with $N_{hp(\tau_1)}$. The time-automata method has a complexity of $O(k^2 H^3_{1^+} N^c_{hp(\tau_1)})$ where $c \geq 3$.

We conducted experiments on over 50 different case-studies, whose parameters are decided as mentioned in Section VI-C. We show the run-time and results of three examples of our experiments in Table III. The second and the third experiments differ with the first experiment in number of tasks and length of hyperperiod of the tasks with higher priorities than the task under firmness analysis, i.e. H_{1^+}. The table confirms that the run-time of the timed-automata model changes drastically with increasing H_{1^+} and number of tasks whereas the run-time of the brute-force search approach is more sensitive to H_{1^+}.

TABLE III: Three examples of the experimental setups considered in this work (time in ms)

Task set	H_{1+}	τ_1		Results		Run time		
		m	k	o_{max}	m_{1-max}	FAn	TA	BF
$\tau_4 = (5, 50, 48, 0), \tau_3 = (7, 50, 47, 12),$ $\tau_2 = (12, 30, 30, 19), \tau_1 = (17, 57, 55, O_1)$	150	155	170	11.88	164	< 0.1s	50s	110s
$\tau_4 = (5, 50, 48, 0), \tau_3 = (7, 47, 46, 12),$ $\tau_2 = (11, 29, 28, 19), \tau_1 = (16, 57, 55, O_1)$	68150	155	170	41530	168	21s	>10h	34000s
$\tau_6 = (5, 50, 58, 0), \tau_5 = (5, 50, 56, 12),$ $\tau_4 = (7, 30, 28, 19), \tau_3 = (7, 50, 50, 37),$ $\tau_2 = (5, 30, 27, 21), \tau_1 = (13, 57, 55, O_1)$	150	155	170	60.27	164	<1s	>10h	182s

VII. Conclusions

The proposed FAn method obtains guaranteed max-min DHs in any window of k consecutive activations of an (m, k)-firm task with unknown first release time in a set of asynchronous tasks running under an SPP policy. We also obtain the first release time for the (m, k)-firm task that results in the max-min DHs. As opposed to traditional schedulability analysis with hard deadlines (which is a special case of firmness analysis), the proposed firmness analysis can be considered as a foundation for scheduling with deadline misses constrained by firmness conditions. Allowing such deadline misses further opens up possibilities for a tighter resource dimensioning. There are two possible future extensions. First, a challenging extension would be to extend the proposed analysis for a general case with more than two synchronous tasks governed by firmness conditions. Second, a thorough investigation on impact of such analysis on the resource dimensioning is another interesting future direction.

Acknowledgment

This research is supported by the ARTEMIS joint undertaking under the ALMARVI project (621439).

References

[1] G. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011.

[2] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *Computers, IEEE Transactions on*, vol. 44, no. 12, pp. 1443–1451, 1995.

[3] Autosar automotive open system architecture (2016). [Online]. Available: http://www.autosar.org/

[4] M. Coutinho, J. Rufino, and C. Almeida, "Response time analysis of asynchronous periodic and sporadic tasks sheduled by a fixed priority preemptive algorithm," in *Real-Time Systems (ECRTS), Euromicro Conference on*. IEEE, 2008 Czech Republic.

[5] G. Bernat, A. Burns, and A. Llamosi, "Weakly hard real-time systems," *Computers, IEEE Transactions on*, vol. 50, no. 4, pp. 308–321, 2001.

[6] G. Bernat, *Specification and analysis of weakly hard real-time systems*. Universitat de les Illes Balears, Spain, 1998.

[7] G. Behrmann, A. David, K. Larsen, J. Hakansson, P. Petterson, W. Yi, and M. Hendriks, "Uppaal 4.0," in *Quantitative Evaluation of Systems, Third International Conference on*. IEEE, 2006 California.

[8] E. van Horssen, A. Behrouzian, D. Goswami, D. Antunes, T. Basten, and M. Heemels, "Performance analysis and controller improvement for linear systems with (m, k)-firm data losses," in *European Control Conference (ECC)*. IEEE, 2016 Denmark.

[9] F. Felicioni, N. Jia, Y. Song, and F. Simonot-Lion, "Impact of a (m, k)-firm data dropouts policy on the quality of control," in *Factory Communication Systems, Workshop on*. IEEE, 2006 Italy.

[10] J. Ning, Y.-Q. Song, and L. Rui-Zhong, "Analysis of networked control system with packet drops governed by (m, k)-firm constraint," in *Fieldbus Systems and Their Applications, International Conference on*. IFAC, 2005 Mexico.

[11] H. Ismail, D. Jawawi, and M. Isa, "A weakly hard real-time tasks on global scheduling of multiprocessor systems," in *Software Engineering Conference (MySEC)*. IEEE, 2015 Malaysian.

[12] Y. Kong and H. Cho, "Guaranteed scheduling for (m, k)-firm deadline-constrained real-time tasks on multiprocessors," in *Parallel and Distributed Computing, Applications and Technologies (PDCAT), International Conference on*. IEEE, 2011 South Korea.

[13] R. West and C. Poellabauer, "Analysis of a window-constrained scheduler for real-time and best-effort packet streams," in *Real-Time Systems Symposium, International conference on*. IEEE, 2000 Florida.

[14] S. Tobuschat, R. Ernst, A. Hamann, and D. Ziegenbein, "System-level timing feasibility test for cyber-physical automotive systems," in *Industrial Embedded Systems (SIES), International symposium on*. IEEE, 2016 Poland.

[15] A. R. Behrouzian, D. Goswami, M. Geilen, M. Hendriks, H. A. Ara, E. van Horssen, W. Heemels, and T. Basten, "Sample-drop firmness analysis of tdma-scheduled control applications," in *Industrial Embedded Systems (SIES), International Symposium on*. IEEE, 2016 Poland.

[16] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[17] D. Borwein and J. Borwein, "Fixed point iterations for real functions," *Journal of Mathematical Analysis and Applications*, vol. 157, no. 1, pp. 112–126, 1991.

[18] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[19] M. Jaghoori, F. de Boer, T. Chothia, and M. Sirjani, "Schedulability of asynchronous real-time concurrent objects," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 402–416, 2009.

[20] F. Cassez and K. Larsen, "The impressive power of stopwatches," in *Concurrency Theory, International Conference on*. Springer, 2000 United Kingdom.