Handling Transients of Dynamic Real-Time Workload Under EDF Scheduling

Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo, *Fellow, IEEE*

Abstract—Real-time dynamic workload consists of tasks that can arbitrarily join and leave the system at run-time. To avoid incurring deadline misses, tasks that request to join the system must pass an admission test, which has to cope with potential scheduling transients originated by the residual effect of the tasks that previously left the system. This phenomenon may require some tasks to suffer an admission delay before being accepted for execution. This paper focuses on uniprocessor earliest-deadline first (EDF) scheduling with constrained deadlines and explicitly considers methods for handling scheduling transients in the presence of dynamic real-time workload. A generalized analysis framework is first presented to overcome several limitations of the existing approaches (including the support for overlapping transients), and is then used to derive methods for computing bounds on the admission delays incurred by tasks. Building on such results, an on-line protocol is proposed to handle the admission control of a dynamic workload, which also comes with a variant that can execute in polynomial time to favor its practical application. Furthermore, the paper shows how the presented analysis can be used off-line for analyzing mode-changes among static task sets. Experimental results are finally presented to evaluate the proposed algorithms.

Index Terms—Real-time systems, scheduling transients, dynamic workload, mode change, schedulability analysis, deadline-based scheduling

1 Introduction

Several real-time applications are characterized by a dynamic workload, where computational activities (tasks) are allowed to exit and join the system at run-time. Some representative examples of such applications are multimedia software systems [1], cloud databases [2], and open environments, in which new software components may arrive in the system while other components are already executing. Most of the operating systems that include real-time scheduling capabilities are conceived to support a dynamic workload, as tasks can freely be created and destroyed at run-time. For instance, this is the case for VxWorks, QNX, and Linux.

To avoid incurring deadline misses, tasks that request to join the system must pass an *admission test* before being admitted for execution, which generally consists in a schedulability test. However, when a new task requests to join the system after other tasks left, the system can experience a *scheduling transient* originated by the residual effect of the leaving tasks. That is, despite the system would result schedulable in steady-state conditions according to a classical schedulability test after admitting the new task, the execution of the leaving tasks may already have influenced the schedule of the system so that the admission of a new task can generate deadline misses. Therefore, as identified in [3], [4], a standard steady-state schedulability analysis that neglects the leaving tasks would not be safe, if deadline misses cannot be tolerated. A solution to this problem consists in *delaying* the admission of new tasks up to a safe time in which they can start executing without causing (or incurring) deadline misses. The time a task waits during a scheduling transient is denoted as *admission delay*.

The analysis of scheduling transients is similar to the one of systems with mode-changes, which has been extensively studied in the literature (a detailed review is reported in Section 2). However, dealing with a dynamic workload presents considerable differences. Two of the most important ones are the need for computing admission delays *online*, and the possibility to manage *overlapping transients* originated by multiple (and potentially interleaved) leaving/joining tasks; i.e., the case in which multiple tasks request to join the system at different times while the system is still experiencing a scheduling transient.

To the best of our knowledge, the methods addressing such problems under earliest-deadline first (EDF) scheduling are only available for implicit-deadline tasks, where the analysis is performed with simple utilization-based tests. Besides the theoretical relevance of supporting constrained deadlines, recent works showed that EDF-based semi-partitioned scheduling allows achieving very high schedulability performance with limited run-time overhead, but the methods require dealing with constrained deadlines even to schedule implicit-deadline tasks [5], [6]. Hence, this work can also serve as a building block for future work targeting the computation of admission delays under semi-partitioned EDF-based scheduling.

- *The authors are with the TeCIP Institute of the Scuola Superiore SantAnna, Pisa 56124, Italy.*
  E-mail: {daniel.casini, alessandro.biondi, giorgio.buttazzo}@santannapisa.it.

Manuscript received 17 Jan. 2018; revised 3 Oct. 2018; accepted 15 Nov. 2018. Date of publication 27 Nov. 2018; date of current version 16 May 2019. (Corresponding author: Daniel Casini.)
Recommended for acceptance by I. Puaut.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2018.2882451

Contributions. This paper focuses on uniprocessor (or multiprocessor partitioned) EDF scheduling with constrained deadlines and makes the following three contributions:

- First, a schedulability analysis framework is presented to cope with overlapping scheduling transients and constrained-deadline tasks, which is entirely based on a new modeling approach that has been conceived to support a dynamic real-time workload.
- Second, based on the proposed analysis, an on-line protocol is proposed to handle the admission control of a dynamic workload. The proposed protocol comes with two algorithms for computing the admission delays of tasks, where one of them has a polynomial-time complexity to favor its online usage.
- Third, the paper shows how the presented analysis can be used off-line for guaranteeing the schedulability of transients originated by mode changes.

Experimental results are finally presented to assess the performance of the proposed approaches and compare them with state-of-the-art methods to deal with mode-changes.

Paper Structure. The remainder of this paper is organized as follows. Section 2 provides a state-of-the-art analysis for the related work, illustrating the differences with the present paper. Section 3 presents the system model and recalls some essential background. Section 4 presents two schedulability analysis techniques for managing scheduling transients. Section 5 proposes an on-line protocol to handle transients and two algorithms for computing the admission delays of tasks. Section 6 discusses how the presented results can also be adopted to analyze mode changes of static task sets. Section 7 reports an experimental study that has been conducted to assess the performance of the proposed approaches. Section 8 concludes the paper and states future work.

2 Related Work

Methods for dealing with scheduling transients have been extensively studied in the literature. Many authors analyzed this issue in the context of multi-moded systems, in which the application can switch among different operating modes, each corresponding to a different task set. Other authors explicitly targeted dynamic workloads, but focused on implicit-deadline tasks or soft real-time guarantees (e.g., bounded tardiness).

Mode-Change. Although some solutions developed for analyzing mode changes could be extended for being applied to the problem addressed in this paper, all of them do not support some peculiar features that are needed to handle the admission control of real-time workload. In particular, most of the works targeting mode changes generally assume that the operating modes of an application are known a-priori, and hence target static real-time workload whose schedulability can be tested off-line for a *given*transition delay. Conversely, in the presence of a dynamic workload, operating modes are not known a priori and admission delays must be computed on line. In addition, new tasks may request to join the system while other tasks are still waiting for being admitted, thus originating *overlapping transients*.

Unfortunately, to the best of our knowledge, all the works focused on mode changes do not manage overlapping transients, since they assume that a mode change can only occur in steady-state conditions and not during another transient. Furthermore, since the admission control of dynamic workloads must be managed at run time, it is possible to exploit on-line scheduling information to compute smaller admission delays. Clearly, this cannot be done for protocols developed for managing mode changes in static task sets.

One of the first works targeting mode changes is due to Fohler [7], who targeted table-driven scheduling of real-time tasks. In the context of uniprocessor fixed-priority (FP) scheduling, several protocols and analysis techniques for mode-changes have been proposed. The interested reader can refer to the survey by Real and Crespo [18]. Andersson [10] focused on EDF scheduling and analyzed the mode-change protocol proposed by Sha et al. [19] but only assuming implicit-deadline tasks. This assumption allowed the author to prove an utilization bound. Phan et al. [13] proposed a multi-mode automata model and the related compositional analysis technique for processing multiple event streams. Their analysis considers overlapping mode changes, but it requires exploring a reachability graph with an algorithm that has exponential complexity.

Fisher and Ahmed [14] proposed two sufficient schedulability tests for EDF with constrained deadlines considering applications running under temporal isolation (e.g., scheduled by a reservation server) that can experience mode changes. The two proposed approaches differ in the assumptions: in the first one, the authors considered sequences of mode-changes that are fixed a-priori, while in the second one the system can arbitrarily switch between a *given* set of modes. Both the approaches assume a *given* transition delay between the modes and allow for overlapping mode changes. Later, Ahmed and Fisher [20] developed a parallel algorithm to speed up the schedulability tests developed in [14]. Stoimenov et al. [15] addressed the problem of mode changes under both EDF and FP scheduling considering the general event stream model and applying real-time calculus to analyze the system. However, the authors formulated the analysis in the continuous domain, without providing bounds on the analysis interval, hence the method cannot be used to implement a schedulability test. Similarly, a binary search over an unbounded domain is briefly suggested for computing transition delays between operating modes. In a later work, Stoimenov et al. [21] considered the mode-change problem in the context adaptive reservation servers using time division multiple access (TDMA), investigating on the resource provisioning during mode switches. Santinelli et al. [22] proposed schedulability analysis for multi-moded resource reservation servers. The authors proposed a framework to deal with both *inter-server* and *intra-server* schedulability analysis; however, the contribution of the paper regards only *intra-server* analysis, whereas inter-server schedulability is addressed by means of the results of other works (e.g., [3], [8]). Other works addressed mode changes under multiprocessor global scheduling. Rattanatamrong and Fortes [23] proposed a global real-time multiprocessor scheduling algorithm managing mode transitions, called `EAGLE-T`. In their algorithm tasks are characterized by different utilizations in different modes, and adapt their utilization when mode changes occur. The authors focused on implicit-deadline tasks only.

Nelis et al. [11] proposed two algorithms, named AM-SO and SM-SO (working under different assumptions), to

TABLE 1
Comparison of the Related Work

Paper	Scheduler	Deadline	Processors	Workload	Approach	Overlapping transients
Fohler [7]	Pre run-time	Any	UP	Static	Table Driven	NO
Buttazzo et al. [3]	EDF	Implicit	UP	Dynamic	Online protocol	YES
Guangming [8]	EDF	Implicit	UP	Dynamic	Online protocol	YES
Rattanatamrong and Fortes [9]	Custom	Implicit	MP-G	Dynamic	Online protocol	NO
Andersson [10]	EDF	Implicit	UP	Static	Analysis	NO
Nelis et al. 2009 [11]	EDF / FP	Constrained	MP-G	Static	Online protocol	NO
Nelis et al. 2011 [12]	EDF	Constrained	MP-G	Static	Online protocol	NO
Phan et al. [13]	EDF	Arbitrary	UP	Static	Analysis	YES
Fisher et al. [14]	EDF	Constrained	UP	Static	Analysis	YES
Stoimenov et al. [15]	EDF/FP	Arbitrary	UP	Static	Analysis	NO
Lee and Shin [16]	EDF/FP	Constrained	MP-G	Static	Analysis	NO
Block and Anderson[17]	Custom	Implicit	UP	Dynamic	Online Protocol	NO
This Paper	***EDF***	Constrained	***UP***	***Dynamic and Static***	***Analysis and Online Protocol***	***YES***

handle transitions among modes that can be used with any global preemptive job-level fixed-priority scheduling algorithm. In a later work, Nelis et al. [12] proposed another mode-change protocol designed to work in conjunction with global EDF (G-EDF) schedulers, ensuring that (i) every job completes within its absolute deadline and (ii) during a mode-change new mode tasks are activated within their *relative transition-deadline*.

Lee and Shin [16] extended a schedulability test proposed by Bertogna et al. [24] for both G-EDF and global fixed-priority (G-FP) to cope with the off-line analysis of mode changes consisting of a *single* transition. No transition delay has been considered in the analysis. The authors also provided an approach for computing a transition order for tasks that allow limiting the amount of interference generated during the mode change.

Finally, some authors also addressed the mode-change problem in the field of real-time networks: most relevant to us are the works by Kopetz et al. [25] and Heilmann et al. [26] .

Dynamic Real-Time Workload. Under EDF, a solution to handle scheduling transients in the presence of dynamic workloads has been presented by Buttazzo et al. [3], who proposed an elastic scheduling framework where a resource manager can modify the period of the tasks. Whenever the period of a task is modified, the system can incur in a transient and the modified task may suffer an admission delay. Bounds for such a delay are computed in [3], even in the presence of overlapping transients; however, the work is limited to implicit-deadline tasks. The latter assumption allowed the authors to take advantage of simple utilization-based tests for analyzing the system and derive bounds on the admission delays.

Guangming [8] improved the result derived in [3] by providing a tighter bound on the admission delay, but again the solution is valid only for tasks with implicit deadline. Block and Anderson [17] and Block et al. [27] proposed a task reweighting scheme for implicit-deadline tasks working under partitioned and P-Fair scheduling, respectively. The authors bounded the delay that can be experienced by a task when a reweighting event occurs. Andersson and Ekelin [28] proposed an admission controller for task sets composed of aperiodic and periodic tasks. The controller exploits the fact that the release times of periodic tasks is known a priori. No admission delays were considered.

The problem of mode-changes and admission control have been more recently addressed in the context of mixed-criticality scheduling with the Vestal's model, where a mode-change occurs when the system switches its criticality level. For instance, Masrur et al. [29] extended the EDF-VD [30] algorithm with a bi-level deadline assignment, which handles a potential increase of the workload due to dynamic task arrivals (or criticality mode changes) by assigning smaller virtual deadlines to high-criticality tasks. Gu and Easwaran [31] improved the schedulability of EDF-VD by proposing an alternative test based on demand bound functions.

Comparison and Discussion. To better illustrate the differences of the present paper with respect to the related work and to position this paper in the literature, Table 1 presents a taxonomy organized according to the following characteristics (reported in the table columns): (i) scheduler type (e.g., EDF or fixed-priority); (ii) type of deadline (implicit, constrained, or arbitrary); (iii) single-processor (or partitioned multi-processor) or multi-processor (global); (iv) requirement of a-priori knowledge of the workload (static vs. dynamic workload); (v) type of the adopted approach (mainly distinguished between schedulability analysis and online protocol); (vi) ability to handle overlapping transients. For the sake of clarity, only the works most relevant to us have been included in Table 1.

Finally, it is worth noting that it is a common belief that the duration of a scheduling transient is bounded by the deadline of the task that left the system, which hence would provide a trivial upper-bound for the admission delay of future tasks. However, this property only holds for implicit-deadline tasks.[1] Fig. 1 shows an example of a schedule that proves that this property is not true in the presence of constrained-deadline tasks. In the figure, τ_1 quits the system at time $t = 10$, and the new task τ_4 arrives exactly in correspondence of the deadline of τ_1, i.e., at time $t = 20$. However, although both task sets $\{\tau_1, \tau_2, \tau_3\}$ and $\{\tau_1, \tau_2, \tau_4\}$ are schedulable (this can be verified by applying the well-established EDF analysis proposed by Baruah et al. [32]), τ_4 experiences a deadline miss.

1 The validity of this result under implicit-deadline can be easily proved by upper-bounding the formula for the transient delay proposed in [3] with the relative deadline.

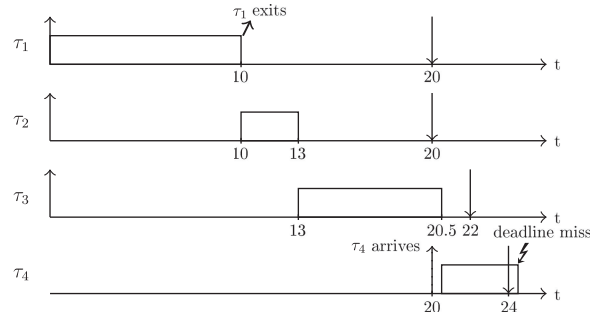

Fig. 1. Example in which the scheduling transient originated by a task (τ_1 in the figure) that leaves the system is not exhausted at the task's deadline. The parameter of the tasks in the example (described by worst-case computation time, relative deadline, and period) are: $\tau_1 = \langle 10, 20, 20\rangle$, $\tau_2 = \langle 3, 20, 20\rangle$, $\tau_3 = \langle 7.5, 22, 22\rangle$, $\tau_4 = \langle 4, 4, 20\rangle$. Task τ_1 leaves the system at time $t = 10$, while task τ_4 joins the system at time $t = 20$, equal to τ_1's deadline.

3 System Model and Background

This paper considers a uniprocessor system that executes a dynamic real-time workload consisting of sporadic real-time tasks that can arbitrarily join or leave the system at run-time. Tasks are managed under preemptive EDF scheduling. Each task is characterized by a *worst-case execution time* (WCET) C_i, a minimum inter-arrival time T_i, and a relative deadline $D_i \leq T_i$. The utilization of a task is denoted as $U_i = C_i/T_i$. Before being admitted for execution, a task that requests to join the system must pass an *admission test* based on its parameters (the WCET can be enforced with a budgeting mechanism, e.g., as available in Linux with the `SCHED_DEADLINE` scheduling class [33] or in AUTOSAR [34]).

In order to analyze scheduling transients, this paper considers scenarios in which the system is subject to a *sequence of events* $S = \{E_1, E_2, \ldots, E_N\}$, where each event $E_k = \langle \tau_i, t_k, type\rangle$ is characterized by

- a task τ_i;
- a time t_k, relative to the beginning of the last *busy period*, in which the event is occurred; and
- the type of the event, where *type* = {`ARRIVAL, EXIT`} represents the arrival or the exit of τ_i, respectively.

A *busy period* is defined as a time interval such that (i) the processor is busy at all times during the interval, (ii) just before the interval, the processor is idle, and (iii) just after the interval, the processor is idle. The system is said to be idle before its startup. All the times are assumed to be relative to the beginning of an arbitrary busy period of interest, which is studied by means of the analysis techniques presented in this paper: this is because, once an idle time occurs, all the scheduling transients are exhausted [35].

To simplify the notation, it is assumed that a task can have at most one arrival and one exit event in a sequence S, i.e., once a task τ_i leaves the system it cannot request to join the system again. Note that multiple join requests of a task τ_i can easily be handled by considering the arrival of different tasks with the same parameters of τ_i. The set of tasks for which there exists an exit event into the sequence S is denoted by $\Gamma^e(S)$. Similarly, $\Gamma^a(S)$ denotes the set of tasks for which there exists an arrival event in S. Given a task $\tau_i \in \Gamma^e(S)$, t_i^e is defined as the time at which the task left the system, i.e., $t_i^e = t_k : \exists E_k = \langle \tau_i, t_k, \texttt{EXIT}\rangle \in S$. In a similar way, t_i^a denotes the time at which task $\tau_i \in \Gamma^a(S)$ arrived in the system, i.e., $t_i^a = t_k : \exists E_k = \langle \tau_i, t_k, \texttt{ARRIVAL}\rangle \in S$. For each task $\tau_i \in \Gamma^a(S)$, the *admission delay* $\lambda_i \geq 0$ is defined such that $\Delta_i = t_i^a + \lambda_i$ is the actual time at which τ_i is admitted for execution. Once a task is admitted, it can start releasing jobs following a sporadic pattern that respects its minimum inter-arrival time. Any non-admitted task is rejected. For consistency, we require that if there exists a task τ_i such that $\Delta_i > 0$ and $\tau_i \in \Gamma^e(S)$, then $\Delta_i \leq t_i^e$, i.e., a task can exit only after the time it actually joined the system. Note that, differently from other proposals, this model does not forbid overlapping scheduling transients.

TABLE 2
Main Notation Adopted throughout the Paper

Symbol	Description
τ_i	i^{th} task
C_i	worst-case execution time of τ_i
T_i	minimum inter-arrival time of τ_i
D_i	relative deadline of τ_i
U_i	utilization of τ_i
t_i^e	time in which τ_i lefts the system
t_i^a	time in which τ_i requests to join the system
λ_i	admission delay of τ_i
Δ_i	time in which τ_i actually enters the system ($\Delta_i = t_i^a + \lambda_i$)
S	sequence of events
E_k	specific event $E_k = \{\tau_i, t_k, \texttt{ARRIVAL}\}$
Γ_O	set of tasks in the system not interested by sequences
$\Gamma^a(S)$	set of tasks admitted during the sequence S
$\Gamma^e(S)$	set of tasks that left the system during the sequence S
$\Gamma_F(S)$	set of tasks in the system at the end of the sequence S
$dbf_i(t)$	demand bound function of τ_i
$dbf_i^e(t, \Delta_i, t_i^e)$	demand bound function of a task $\tau_i \in \Gamma^e(S)$
$\overline{dbf}_i(t)$	approximate demand bound function of τ_i
$\overline{dbf}_i^e(t, \Delta_i, t_i^e)$	approximate demand bound function of $\tau_i \in \Gamma^e(S)$
$\mathcal{D}(\tau_i)$	set of check-point for τ_i according to Theorem 2
$\xi(\tau_i)$	set of check-point for τ_i according to Theorem 4

For the sake of completeness, the analysis presented in this paper also considers a task set Γ_O that was admitted for execution *before* the beginning of a given sequence S and that is not interested by the events in S, i.e., none of the tasks in Γ_O leaves the system. All the tasks that do not join the system during a sequence have the admission time set to zero, i.e.,

$$\forall \tau_i \in \Gamma_O \cup \{\Gamma^e(S) \setminus \Gamma^a(S)\},\ \Delta_i = 0. \tag{1}$$

To reduce clutter, the set $\Gamma_F(S) = \Gamma_O \cup \{\Gamma^a(S) \setminus \Gamma^e(S)\}$ is defined, which represents the set of tasks that are present into the system after a sequence S has completed. Finally, the following short notation is adopted: $\lfloor x \rfloor_0 = \max\{0, \lfloor x \rfloor\}$ and $(x)_0 = \max\{0, x\}$. Table 2 summarizes the symbols adopted throughout the paper.

3.1 Background on EDF Analysis

The results presented in this paper build upon the *processor-demand criterion* (PDC) proposed by Baruah et al. [32]. The *demand function* [36] $g_i(t_1, t_2)$ of a task τ_i in an arbitrary interval $[t_1, t_2]$ (with respect to an arbitrary schedule) is defined as the amount of processing time requested by instances (i.e., jobs) of τ_i that have both release times and absolute

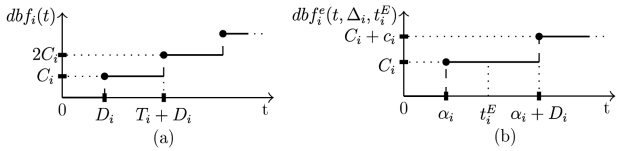

Fig. 2. Illustrations of the demand bound functions used in Section 4 (solid lines). Insets (a) and (b) show functions $dbf_i(t)$ and $dbf_i^e(t, \Delta_i, t_i^e)$, respectively.

deadlines in $[t_1, t_2]$. Formally,

$$g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1 \wedge d_{i,k} \leq t_2} c_{i,k}, \tag{2}$$

where $r_{i,k}$ and $d_{i,k}$ are the release time and the absolute deadline of the k^{th} job of τ_i, respectively, and $c_{i,k} \leq C_i$ is the execution time of the k^{th} job of τ_i.

The PDC is based on the notion of *demand bound function* $dbf(t)$, which provides the maximum demand generated by a task in any interval of length t. Clearly, $dbf(t)$ upper-bounds the demand function, i.e., $g_i(t_1, t_2) \leq dbf_i(t_2 - t_1)$. For a given task τ_i, the demand bound function is defined (see [32]) as:

$$dbf_i(t) = \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor_0 C_i. \tag{3}$$

Such a function is illustrated in Fig. 2a.

Under EDF scheduling, the PDC allows verifying the schedulability of a given set Γ of sporadic tasks with constrained deadlines as stated in the following theorem.

Theorem 1 (Processor Demand Criterion). *A task set Γ of sporadic, arbitrary-deadline tasks is EDF-schedulable if and only if*

$$\forall t \in \mathcal{D}^*, \ \sum_{\tau_i \in \Gamma} dbf_i(t) \leq t, \tag{4}$$

with $\mathcal{D}^* = \bigcup_{\tau_i \in \Gamma} \{t = D_i + fT_i : t < L^* \wedge f \in \mathbb{N}_{\geq 0}\}$, where L^* is the length of the analysis interval (bounds are available in [32], [37]).

4 Transient-Aware Schedulability Analysis

This section presents two analysis techniques that are capable of handling transients under EDF scheduling. As discussed in the previous section, since any transient is exhausted at the first idle time, both the techniques focus on a sequence of events S (arrival or exit of tasks) within a *single* busy-period. For this reason, all the times reported in this section are relative to the start time of the busy-period under analysis.

For the purpose of this section, the time at which the events in S occur and the admission delays of the tasks are assumed to be *given*. The proposed analysis techniques will be then used in the following section as the foundation to develop methods for handling transients on-line, thus computing the admission delays.

The first analysis is presented in Section 4.1 and aims at extending the PDC to verify the system schedulability in the presence of scheduling transients. The second analysis, presented in Section 4.2, is based on an approximation scheme of the PDC and has a polynomial-time complexity.

4.1 PDC-Based Analysis

To begin, it is necessary to extend the definition of demand bound function to cope with tasks $\tau_i \in \Gamma^e(S)$ that leave the system at time t_i^e, that is

$$dbf_i^e(t, \Delta_i, t_i^e) = \begin{cases} dbf_i(t - \Delta_i) & \text{if } t < \alpha_i + D_i, \\ dbf_i(\alpha_i - \Delta_i) + c_i & \text{otherwise,} \end{cases} \tag{5}$$

where

$$\alpha_i = \left\lfloor \frac{t_i^e - \Delta_i}{T_i} \right\rfloor T_i + \Delta_i, \tag{6}$$

and $c_i = min(C_i, t_i^e - \alpha_i)$. Such a function is illustrated in Fig. 2b. Intuitively speaking, α_i is the time at which τ_i releases its *last* job in the scenario in which its demand is maximized, i.e., when jobs are released as soon as possible. As a consequence, function $dbf_i^e(t, \Delta_i, t_i^e)$ exhibits a saturation to $dbf_i(\alpha_i - \Delta_i) + c_i$ after the deadline of such a last job, which occurs at time $\alpha_i + D_i$. The term c_i is provided to bound the time executed by the last job of τ_i in the same worst-case scenario. The following lemma formalizes the validity of function $dbf_i^e(t, \Delta_i, t_i^e)$.

Lemma 1. *For a task $\tau_i \in \Gamma^e(S)$, the following inequality holds:*

$$\forall t \geq 0, \quad g_i(0, t) \leq dbf_i^e(t, \Delta_i, t_i^e). \tag{7}$$

Proof. Without loss of generality, let us assume a busy-period starting at time $t = 0$ and let α' be the time in which the last job of τ_i is released. Note that the busy-period includes jobs of τ_i only up to time t_i^e. By looking at the definition of demand function in Equation (2), task τ_i can then contribute with demand in the busy-period of interest only up to time $\alpha' + D_i$, i.e., the absolute deadline of its last job. Consequently, if $t < \alpha' + D_i$, the processing time demanded by τ_i in $[0, t]$ is given by $g_i(\Delta_i, t)$, which is upper-bounded by $dbf_i(t - \Delta_i)$.

For $t \geq \alpha' + D_i$, τ_i can contribute with processing demand only with its last job, then no further demand contribution is possible. Let c' be the execution time of τ_i's last job. Hence, the demand generated in $[0, t]$ is constant and equal to $g_i(\Delta_i, \alpha') + c'$, where the first term accounts for the demand generated by all the jobs of τ_i except the last one. Such a contribution is upper-bounded by $dbf_i(\alpha' - \Delta_i) + c'$.

As stated in the standard PDC-based analysis of EDF scheduling, the demand of a sporadic task is maximized when jobs are released as soon as possible. Under this scenario, the last job of τ_i is released at time $\alpha_i = \lfloor (t_i^e - \Delta_i)/T_i \rfloor T_i + \Delta_i$. The lemma follows by replacing $\alpha' = \alpha_i$ and noting that the execution time of the last job of τ_i is bounded by the minimum between the time span that goes from the release of the last job and the exit time, i.e., $t_i^e - \alpha_i$, and the WCET C_i. □

With the above lemma in place, it is possible to present another key lemma that expresses a condition under which a system that is subject to the sequence S is schedulable even in the presence of transients.

Lemma 2. *A system that is subject to a sequence of events S does not incur in deadline misses under EDF scheduling if*

$$\forall t \geq 0, \sum_{\tau_i \in \Gamma_F(S)} dbf_i(t - \Delta_i) + \sum_{\tau_i \in \Gamma^e(S)} dbf_i^e(t, \Delta_i, t_i^e) \leq t. \quad (8)$$

Proof. Without loss of generality, assume a busy-period starting at time $t = 0$. Due to the optimality of EDF [32], the system is schedulable if for all intervals $[0, t]$, with $t \geq 0$, the overall processing time demanded to the system never exceeds the interval length t. Consider an arbitrary EDF schedule in $[0, t]$. We proceed by considering the individual contribution to the overall processing time demanded by each task interested by the sequence S or present into the set Γ_O.

The processing time demanded in $[0, t]$ by a task $\tau_i \in \Gamma_O$ (i.e., not interested by the events in the sequence S) is given by $g_i(0, t)$. Such a function is upper-bounded by $dbf_i(t)$ (note that $\Delta_i = 0$ for such tasks).

A task $\tau_i \in \Gamma^a(S)$ that joins the system is admitted at time $\Delta_i = t_i^a + \lambda_i$. Hence, the processing time demanded by a task $\tau_i \in \Gamma^a(S) \setminus \Gamma^e(S)$ (i.e., that joins the system and does not leave it later) is given by $g_i(\Delta_i, t)$. Such a function is upper-bounded by $dbf_i(t - \Delta_i)$. By recalling the definition of the set $\Gamma_F(S)$, the latter two contributions are expressed by the first term in Equation (8).

Finally, from Lemma 1, the demand of a task $\tau_i \in \Gamma^e(S)$ can be upper bounded by $dbf_i^e(t, \Delta_i, t_i^e)$. Since Equation (8) accounts for an upper bound on the processing demand imposed by each task within the busy-period of interest, the lemma follows. □

As in the classical PDC analysis, to actually implement a schedulability test based on Lemma 2 we need (i) an upper-bound of the analysis interval, and (ii) a discretization of the analysis interval.

For (i), the following lemma can be used.

Lemma 3. *Let $U_F = \sum_{\tau_i \in \Gamma_F(S)} U_i$. If $U_F < 1$, then Equation (8) is satisfied $\forall t > L^*$, where*

$$L^* = \frac{W_F + W_E}{1 - U_F}, \quad (9)$$

with

$$W_F = \sum_{\tau_i \in \Gamma_F(S)} U_i(T_i - D_i),$$

$$W_E = \sum_{\tau_i \in \Gamma^e(S)} dbf_i(\alpha_i - \Delta_i) + C_i.$$

and α_i is defined as in Equation (6).

Proof. For tasks $\tau_i \in \Gamma^e(S)$, their contribution to Equation (8) can be upper-bounded as $\forall t \geq 0$, $dbf_i^e(t, \Delta_i, t_i^e) \leq dbf_i(\alpha_i - \Delta_i) + C_i$. Hence, their overall contribution can be safely upper-bounded by W_E. For tasks $\tau_i \in \Gamma_F(S)$, their contribution to Equation (8) can be upper-bounded by removing the floor operator from function $dbf_i(t)$, which gives $\forall t \geq 0$, $dbf_i(t - \Delta_i) \leq U_i(t - \Delta_i + T_i - D_i)_0$. Since $\Delta_i \geq 0$, then $\forall t \geq 0$, $U_i(t - \Delta_i + T_i - D_i)_0 \leq U_i(t + T_i - D_i)_0$. Given $D_i \leq T_i$, the $()_0$ operator can be removed from the latter expression. Hence, their overall contribution can be safely upper-bounded by $W_F + tU_F$.

By exploiting such upper-bounds, if $W_E + W_F + tU_F \leq t$ holds for $\forall t > L^*$, then Equation (8) is satisfied $\forall t > L^*$. This upper-bound is a straight line with slope U_F. Since $U_F < 1$, there exists an intersection with the identity function $t = t$. Solving with respect to t, the obtained solution is $t = L^* = (W_F + W_E)/(1 - U_F)$. The lemma follows. □

In the limit case in which $U_F = 1$, the maximum analysis interval can be bounded by the length of the longest busy-period that the system can experience. This case is analogous to the one reported in [37], [38] but considering different arrival curves that can be obtained from the demand bound functions adopted in this section. Details are available in Appendix A.2, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/10.1109/TC.2018.2882451.

Finally, the results of the above lemmas are combined in the following theorem, which expresses a schedulability test.

Theorem 2. *If Equation (8) is verified*

$$\forall t \in \bigcup_{\tau_i \in \Gamma_F(S) \cup \Gamma^e(S)} \mathcal{D}(\tau_i), \quad (10)$$

where

$$\mathcal{D}(\tau_i) = \{t = \Delta_i + D_i + fT_i : t < t^*(\tau_i) \wedge f \in \mathbb{N}_{\geq 0}\},$$

$$t^*(\tau_i) = \begin{cases} L^* & \text{if } \tau_i \in \Gamma_F(S) \\ min(\alpha_i + D_i, L^*) & \text{otherwise,} \end{cases}$$

and α_i defined as in Equation (6), then Equation (8) is verified $\forall t \geq 0$.

Proof. Note that both functions $dbf_i(t - \Delta_i)$ and $dbf_i^e(t, \Delta_i, t_i^e)$ are step-wise monotonic with discontinuities in points $\Delta_i + D_i + fT_i$ with $f \in \mathbb{N}_{\geq 0}$. The theorem follows after recalling Lemma 3, which provides the bound L^*, and that the last discontinuity of function $dbf_i^e(t, \Delta_i, t_i^e)$ occurs for $t = \alpha_i + D_i$. □

The computational complexity of Theorem 2 is the same of the PDC, i.e., pseudo-polynomial if $U < 1$. It is worth mentioning that the presented schedulability test can also be efficiently implemented with the *quick processor-demand analysis* (QPA) algorithm proposed by Zhang and Burns [37].

4.2 FPTAS-Based Analysis

In this section, an approximate analysis for scheduling transients is derived by building upon the *fully polynomial-time approximation scheme* (FPTAS) for the PDC proposed by Fisher et al. [39]. The FPTAS approach is based on approximate demand bound functions that are defined as follows:

$$\overline{dbf}_i(t) = \begin{cases} dbf_i(t) & \text{if } t < \nu_i T_i + D_i \\ C_i + U_i(t - D_i) & \text{otherwise.} \end{cases} \quad (11)$$

As it can be noted from the latter equation, the approximate functions accounts for $\nu_i + 1$ steps (with $\nu_i \in \mathbb{N}_{\geq 0}$) equal to the original demand bound function $dbf_i(t)$, and then use a linear bound with slope U_i for the remaining time intervals. Such an approximate function upper-bounds function $dbf_i(t)$ [39] and is illustrated in Fig. 3a.

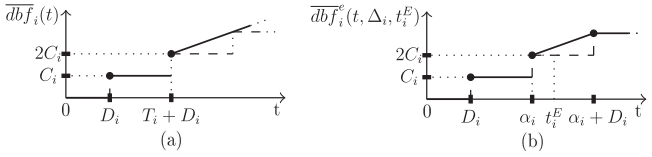

Fig. 3. Illustrations of the approximate demand bound functions used in Section 4 (solid lines). Insets (a) and (b) show the approximate functions $\overline{dbf}_i(t)$ and $\overline{dbf}^e_i(t, \Delta_i, t^e_i)$ respectively. The functions are illustrated for $v_i = 1$ and $\Delta_i = 0$. The dashed lines depict functions $dbf_i(t)$.

Leveraging the definition of $\overline{dbf}_i(t)$, an approximate schedulability test can be formulated to verify the system schedulability in steady-state conditions.

Theorem 3 (From [39]). *A task set Γ of sporadic, constrained-deadline tasks is EDF-schedulable if*

$$\forall t \in \bigcup_{\tau_i \in \Gamma} \overline{\mathcal{D}}(\tau_i),\ \sum_{\tau_i \in \Gamma} \overline{dbf}_i(t) \leq t, \tag{12}$$

with $\overline{\mathcal{D}}(\tau_i) = \{jT_i + D_i,\ j = 0, \ldots, v_i\}$.

As done in the previous section with Equation (5), the approximate demand bound function of a task $\tau_i \in \Gamma^e(S)$ that leaves the system is defined as:

$$\overline{dbf}^e_i(t, \Delta_i, t^e_i) = \begin{cases} \overline{dbf}_i(t - \Delta_i) & \text{if } t < \alpha_i + D_i, \\ \overline{dbf}_i(\alpha_i - \Delta_i) + C_i & \text{otherwise,} \end{cases} \tag{13}$$

where α_i is defined as for Equation (5). The derivation of such a function (illustrated in Fig. 3b) is analogous to Lemma 1 and, by construction, it upper bounds function $dbf^e_i(t, \Delta_i, t^e_i)$.

Leveraging these approximate demand bound functions, it is possible to formulate an approximate version of the schedulability test expressed by Theorem 2.

Theorem 4. *A system that is subject to a sequence of events S does not incur in deadline misses under EDF scheduling if* $\sum_{\tau_i \in \Gamma_F(S)} U_i \leq 1$ *and*

$$\forall t \in \bigcup_{\tau_i \in \Gamma_F(S) \cup \Gamma^e(S)} \xi(\tau_i),$$

$$\sum_{\tau_i \in \Gamma_F(S)} \overline{dbf}_i(t - \Delta_i) + \sum_{\tau_i \in \Gamma^e(S)} \overline{dbf}^e_i(t, \Delta_i, t^e_i) \leq t, \tag{14}$$

where

$$\xi(\tau_i) = \begin{cases} \{\Delta_i + jT_i + D_i\} & \text{if } \tau_i \in \Gamma_F(S),\ j = 0, ..., v_i \\ \{\Delta_i + jT_i + D_i\} & \text{if } \tau_i \in \Gamma^e(S),\ j = 0, ..., v_i \wedge \\ & \Delta_i + jT_i + D_i \leq \alpha_i + D_i, \end{cases}$$

with α_i defined as in Equation (6).

Proof. After recalling Theorem 3 in place of the standard PDC analysis, the proof is analogous to the one of Theorem 2 but considering the *finite* set of discontinuities of the adopted approximate demand bound functions. Such discontinuities occur at points $\Delta_i + jT_i + D_i$ for tasks $\tau_i \in \Gamma_F(S)$. Similarly, for tasks $\tau_i \in \Gamma^e(S)$, the discontinuities occur for the same family of points in time, but are limited by the maximum number of jobs that τ_i can release leaving the system at time t^e_i, which is accounted with the term $\alpha_i + D_i$, as done in Lemma 1. □

The computational complexity of the test provided by Theorem 4 is $\mathcal{O}(|\bigcup_{\tau_i \in \Gamma^*(S)} \xi(\tau_i)|) = \mathcal{O}(\sum_{\tau_i \in \Gamma^*(S)} (v_i + 1))$ where $\Gamma^*(S) = \Gamma_F(S) \cup \Gamma^e(S)$.

5 Handling Transients Online

This section presents an on-line protocol to handle scheduling transients and proposes two methods for computing the admission delays λ_i for tasks that request to join the system. As a prerequisite, the run-time scheduling mechanism is required to keep track of the beginning time of the current busy period, and of the sequence S of events occurred within that interval. The protocol consists of the following rules:

- R1. Whenever a new task τ_i requests to join the system at time t^a_i, the system verifies the *steady-state* schedulability by means of an admission test.
- R2. Whenever there is an idle time, any task that passes the admission test can join the system without incurring an admission delay.
- R3. Whenever a task τ_i passes the admission test, the admission delay λ_i is computed with one of the methods presented in the following sections. The task is admitted for execution at the earliest time between $\Delta_i = t^a_i + \lambda_i$ and the time at which the first idle time occurs.

The admission test used in rule R1 is strictly dependent on the method that is used in rule R3 for computing the admission delays. In the following, building on the analysis techniques presented in Section 4, two methods for computing the admission delays are presented, where each of them is accompanied with a corresponding admission test. The first one is able to compute the minimum delay with respect to the precision enabled by the analysis, but it suffers from a large run-time complexity. The second one is based on an approximation of the analysis and allows computing the admission delays in polynomial time, thus favoring its practical applicability.

5.1 Method 1

This section considers a task τ_N that requests to join a system at time t^a_N and presents a method for computing the admission delay λ_N of τ_N. The system is assumed to be schedulable previously to the arrival of τ_N. Let S be the sequence of events occurred within the current busy period of the system up to time t^a_N, where the last event in S corresponds to the arrival of τ_N. The proposed method assumes that the standard PDC (Theorem 1) is used as admission test, i.e., considering the task set Γ_F in steady-state conditions.

Following the analysis stated by Theorem 2, the objective of the proposed method consists in solving the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & \lambda_N \\ \text{subject to} \quad & \forall t \in \mathcal{D}^*, \\ & \sum_{\tau_i \in \Gamma_F(S) \setminus \{\tau_N\}} dbf_i(t - \Delta_i) + \sum_{\tau_i \in \Gamma^e(S)} dbf^e_i(t, \Delta_i, t^e_i) \\ & \quad + dbf_N(t - t^a_N - \lambda_N) \leq t. \end{aligned} \tag{15}$$

This optimization problem can be solved by applying the following iterative algorithm:

1) Initially, let $\lambda_N = 0$.
2) Apply Theorem 2 to S. If the schedulability test is passed, then τ_N can be admitted with delay λ_N and the algorithm terminates. Otherwise, let $t^* \in \mathcal{D}^*$ be a check-point in which Equation (8) is not satisfied.
3) Compute the minimum delay λ'_N such that the schedulability test in t^* does not fail. Then, set $\lambda_N = \lambda'_N$ and go to step 2.

The validity of this approach is ensured by the fact that all the terms in Equation (8) are monotone in λ_N. In the following, two lemmas are presented to compute the admission delay at step 3. To avoid discussing the limit case of a fully-utilized system, the following results assume $\sum_{\tau_i \in \Gamma_F(S)} U_i < 1$. First note that the failure point t^* can correspond to (i) a check-point in $\mathcal{D}^*$ originated by τ_N, or to (ii) a check-point in $\mathcal{D}^*$ originated by another task $\tau_i \neq \tau_N$. Case (i) is addressed in Lemma 4 and case (ii) in Lemma 5.

Lemma 4. *Let $t^* \in \mathcal{D}(\tau_N)$ be a check-point originated by τ_N in which Theorem 2 is not verified. Also, let $t' = t^* - \lambda_N$. The failure in t^* can be removed by setting the admission delay λ_N of τ_N to the least fixed-point of the following recursive equation:*

$$\lambda'_N = \sum_{\tau_i \in \Gamma_F(S)} dbf_i(t' + \lambda'_N - \Delta_i) + \sum_{\tau_i \in \Gamma^e(S)} dbf^e_i(t' + \lambda'_N, \Delta_i, t^e_i) - t'. \quad (16)$$

Proof. Theorem 2 is verified in the check-point t^* if

$$\sum_{\tau_i \in \Gamma_F(S)} dbf_i(t^* - \Delta_i) + \sum_{\tau_i \in \Gamma^e(S)} dbf^e_i(t^*, \Delta_i, t^e_i) \leq t^*.$$

Being both the demand bound functions involved in the previous inequality non-decreasing in t^*, the minimum admission delay such that the check-point is satisfied is given when the LHS of the inequality is equal to t^*. The resulting equation is recursive, as both sides depend on t^* (which in turn depends on λ_N). The equation can be rewritten as Equation (16) and its convergence (i.e., the existence of a fixed point) can be proved by observing that: (i) it is monotonic non-decreasing as Equation (16) comprises a sum of monotonic non-decreasing functions; and (ii) it is upper-bounded as long as $\sum_{\tau_i \in \Gamma_F(S)} U_i < 1$ (see Appendix A.1, available online, for details). Hence the lemma follows. □

Equation (16) can be solved with a fixed-point iteration starting from $\lambda'_N = \lambda_N$ (i.e., the current admission delay before removing the failure in t^*).

Lemma 5. *Let $t^* \in \mathcal{D}^* \setminus \mathcal{D}(\tau_N)$ be a check-point not originated by τ_N in which Theorem 2 is not verified. The failure in t^* can be removed by setting the admission delay λ_N of τ_N to:*

$$\lambda'_N = t^* - K + T_N - D_N - t^a_N + \epsilon, \quad (17)$$

where $\epsilon > 0$ (arbitrary small) and

$$K = \left(\left\lfloor \frac{t^* - K_F - K_E)}{C_N} \right\rfloor_0 + 1 \right) T_N$$

with

$$K_F = \sum_{\tau_i \in \Gamma_F(S) \setminus \{\tau_N\}} dbf_i(t^* - \Delta_i),$$

$$K_E = \sum_{\tau_i \in \Gamma^e(S)} dbf^e_i(t^*, \Delta_i, t^e_i).$$

Proof. First note that, since t^* does not depend on λ_N, both K_F and K_E are constants. To fix the failure point by setting $\lambda_N = \lambda'_N$, the following condition must be verified:

$$K_F + K_E + dbf_N(t^* - (t^a_N + \lambda_N)) \leq t^*. \quad (18)$$

By replacing the definition of $dbf_N(t^* - (t^a_N + \lambda_N))$ in Equation (18), it follows that

$$\left\lfloor \frac{t^* - (t^a_N + \lambda_N) + T_N - D_N}{T_N} \right\rfloor_0 \leq \frac{t^* - K_F - K_E}{C_N},$$

which, by exploiting the properties of the floor function, can be rewritten as:

$$\frac{t^* - (t^a_N + \lambda_N) + T_N - D_N}{T_N} < \left\lfloor \frac{t^* - K_F - K_E}{C_N} \right\rfloor_0 + 1.$$

Solving with respect to λ_N, the inequality becomes:

$$\lambda_N > t^* - K + T_N - D_N - t^a_N,$$

with K defined as in the lemma statement. Hence the lemma follows. □

Implementation and Complexity. By leveraging some properties of the involved equations, an efficient implementation of the proposed iterative algorithm can be devised. The resulting algorithm is named *ADT* (Admission Delay for Transients) and is reported in Fig. 4. Algorithm *ADT* exploits the observation that, once Equation (15) is satisfied

```
1:  procedure ADT( Γ_F(S), Γ^e(S))
2:      λ_N ← 0;
3:      D^oth ← ⋃_{τ_i ∈ Γ_E(S) ∪ {Γ_F(S) \ {τ_N}}} D(τ_i);
4:
5:      for t ← D^oth ∧ t ≥ t^a_N do
6:          λ_N ← max(λ_N, LEMMA 5 (Γ_F(S), Γ^e(S)));
7:      end for
8:      do
9:          updated ← false
10:         for t ← D(τ_N) do
11:             λ'_N ← LEMMA 4 (Γ_F(S), Γ^e(S));
12:             if (λ'_N > λ_N) then
13:                 updated ← true;
14:                 λ_N = λ'_N;
15:             end if
16:         end for
17:     while (updated==true);
18:     return λ_N;
19: end procedure
```

Fig. 4. Pseudo-code for the ADT (Admission Delay for Transients) algorithm.

in a check-point $t^* \in \mathcal{D}^* \setminus \mathcal{D}(\tau_N)$ by using a particular delay λ_N, it remains satisfied $\forall \lambda'_N \geq \lambda_N$. This property directly follows from the monotonicity in λ_N of the demand bound function of τ_N. The main advantage introduced by the latter result is that it is not necessary to verify Theorem 2 many times, but instead it is possible to design an algorithm that requires to process each check-point in the set $\mathcal{D}^* \setminus \mathcal{D}(\tau_N)$ only once.

The ADT algorithm starts by initializing $\lambda_N = 0$ and by setting D^{oth} as the set of check-points that are not originated by τ_N. Then, it proceeds by processing such check-points by means of Lemma 5, updating λ_N accordingly (line 6). Note that the algorithm avoids processing the check-points before time t_N^a at which τ_N requested to join the system: this is because the demand contribution of τ_N cannot affect them. Finally, the algorithm processes the check-points originated by τ_N (set $\mathcal{D}(\tau_N)$) by means of Lemma 4. Such check-points are processed as long as λ_N does not increase (see the loop at lines 8-17). The termination of the algorithm is guaranteed by the fact that λ_N is never decreased and that λ_N is upper-bounded by a constant term as long as $\sum_{\tau_i \in \Gamma_F(S)} U_i < 1$ (see Appendix A.1, available online, for details).

As for the standard PDC analysis, the *ADT* algorithm runs for a pseudo-polynomial number of iterations as long as the steady-utilization of the system is strictly less than one. Each iteration has either (i) a constant-time complexity, when Lemma 4 is applied (e.g., by storing the sum of demand-bound functions in an incremental fashion whenever a new task joins the system), or (ii) a $\mathcal{O}(\lambda_N^*)$ complexity when Lemma 5 is used, where λ_N^* is an upper-bound on λ_N (as discussed after the lemma). The average-case performance of the algorithm can be further improved by computing the bound L^* (see Lemma 3) at each iteration in which λ_N changes. Since L^* decreases as λ_N increases, using an updated value of L^* may allow to processing a lower number of check-points.

5.2 Method 2: Approximated Delay Computation

This section aims at deriving an approximate algorithm to compute the admission delay in polynomial time. To this end, the FPTAS-based analysis introduced in Section 4.2 is used as admission test in rule R1 of the proposed protocol.

As done in the previous section, consider a task τ_N for which it is required to compute a safe admission delay λ_N, and let S be the sequence of events occurred within the current busy-period of the system up to the arrival of τ_N (time t_N^a). The same assumptions stated in the previous section have been adopted, with the only exception being that Theorem 3 is used to verify the steady-state schedulability. Our objective is to leverage Theorem 4 to express a transient-aware schedulability test in the form of H constraints $\lambda_N \geq V_z$, with $z = 1, \ldots, H$, where V_z is a *constant* term. Once this has been accomplished, a safe admission delay can then be computed as

$$\lambda_N = max_{z=1,\ldots,H}\{V_z\}.$$

More specifically, each term V_z will be derived by the check-points of the test in Theorem 4, for a total of $H = |\{t \in \cup_{\tau_i \in \Gamma^e(S) \cup \Gamma_F(S)} \xi(\tau_i) : t \geq t_N^a\}|$ constraints.

In the following, four lemmas will be presented to cope with the derivation of the terms V_z from the check-points. Note that each check-point can be originated by (i) τ_N, or (ii) another task $\tau_i \neq \tau_N$.

Case (i). Following Theorem 4, a check-point originated by τ_N can be expressed as $t^* = t_N^a + \lambda_N + D_N + jT_N$ (with $j \in \mathbb{N}_{\geq 0}$). Due to the dependency of t^* on λ_N, a circular dependency arises when evaluating in t^* the demand bound functions of the other tasks $\neq \tau_N$. In fact, being λ_N the parameter to be computed (and hence unknown), such demand bound functions can be either in their piece-wise constant part or in their linear part; hence, their expression is not known a-priori. The following lemma allows breaking such a circular dependency by deriving a safe lower bound for λ_N.

Lemma 6. *Let $U_F = \sum_{\tau_i \in \Gamma_F(S) \setminus \{\tau_N\}} U_i$ and let $t^* \in \xi(\tau_N)$ be a check-point of Theorem 4 originated by τ_N defined as $t^* = \Delta_N + t'$ with $t' = D_N + jT_N$ and $j \in \mathbb{N}_{\geq 0}$. The check-point t^* is verified if*

$$\lambda_N \geq \frac{\overline{dbf}_N(t') + \overline{K}_F + \overline{K}_E - t_N^a - t'}{1 - U_F}, \tag{19}$$

where

$$\overline{K}_F = \sum_{\tau_i \in \Gamma_F(S) \setminus \{\tau_N\}} C_i + U_i(t_N^a + t' - \Delta_i - D_i)_0,$$

$$\overline{K}_E = \sum_{\tau_i \in \Gamma^e(S)} \overline{dbf}_i(\alpha_i - \Delta_i) + C_i,$$

and α_i is defined as in Equation (6).

Proof. The lemma follows by exploiting upper bounds on the terms that compose Equation (14) and the fact that such terms are non-decreasing in t. In particular, the upper bound on functions $\overline{dbf}_i(t - \Delta_i)$ is obtained by degenerating the approximation to a single discontinuity point, thus obtaining a linear and continuous bound equal to $C_i + U_i(t - \Delta_i - D_i)_0$. By replacing $t = t^*$, such an upper bound can be rewritten as $C_i + U_i(t_i^a + t' - \Delta_i - D_i)_0 + U_i\lambda_N$. Then, summing over all tasks in $\Gamma_F(S)$, the upper bound on the first term of Equation (14) results $\overline{K}_F + U_F\lambda_N + \overline{dbf}_N(t')$. Note that there is no need to leverage an upper bound on function $\overline{dbf}_N(t)$, as in the check-point t^* its value does not depend on λ_N (i.e., $\overline{dbf}_N(t^* - \Delta_N) = \overline{dbf}_N(t')$), hence $\overline{dbf}_N(t')$ is a constant term. Finally, the upper bound on functions $\overline{dbf}_i^e(t, \Delta_i, t_i^e)$ is obtained by noting that such functions exhibit a saturation to $\overline{dbf}_i(\alpha_i - \Delta_i) + C_i$. □

The upper bound provided by the above lemma can be quite coarse, especially for high values of U_F; however, it can be used as an initial solution to devise an iterative method that refines the bound on λ_N. In fact, by leveraging a given upper bound λ_N^{ub} on λ_N, the following lemma allows tightening the admission delay.

Lemma 7. *Let $t^* \in \xi(\tau_N)$ be a check-point of Theorem 4 originated by τ_N defined as $t^* = \Delta_N + t'$ with $t' = D_N + jT_N$ ($j \in \mathbb{N}_{\geq 0}$). Given an upper bound $\lambda_N^{ub} \geq \lambda_N$, the check-point is verified if*

$$\lambda_N \geq \overline{dbf}_N(t') + \overline{K} - t_N^a - t', \tag{20}$$

with

$$\overline{K} = \sum_{\tau_i \in \Gamma_F(S) \setminus \{\tau_N\}} \overline{dbf}_i(t'' - \Delta_i) + \sum_{\tau_i \in \Gamma^e(S)} \overline{dbf}_i^e(t'', \Delta_i, t_i^e),$$

where $t'' = t_N^a + \lambda_N^{ub} + t'$.

Proof. As done in the proof of Lemma 6, the lemma follows by exploiting upper bounds on the terms that compose Equation (14) and the fact that such terms are non-decreasing in t. Since $\lambda_N^{ub} \geq \lambda_N$, then $t'' \geq t^*$. As a consequence, $\overline{dbf}_N(t') + \overline{K}$ provides a safe upper bound of the LHS of Equation (14) when evaluated in the check-point $t = t^*$. □

Lemma 7 can be repeatedly applied, thus generating a non-increasing sequence of lower bounds on λ_N. At the first iteration, λ_N^{ub} is set at the value obtained with Lemma 6; then, the obtained bound can in turn be used to set λ_N^{ub} for a next application of Lemma 7, and so on for a desired number of iterations.

Case (ii). Now, consider the case in which a constraint in the admission delay λ_N is derived from check-point of Theorem 4 that is *not* originated by τ_N. In this case, being the check-point independent of λ_N, all the terms in Equation (14) are constant with the exception of $\overline{dbf}_N(t - \Delta_N)$, which instead depends on λ_N through Δ_N. Since function $\overline{dbf}_N(t - \Delta_N)$ is composed of a piece-wise constant part and a linear part, two corresponding sub-cases must be considered. These are respectively managed by Lemma 8 and Lemma 9, which provide *mutually-exclusive* conditions (note the second equations in the systems considered by the two lemmas).

Lemma 8. *Let $t^* \in \xi(\tau_i)$ be a check-point of Theorem 4 originated by $\tau_i \neq \tau_N$. The check-point is verified if the following system of equations is verified:*

$$\begin{cases} \lambda_N > t^* - \overline{K} + T_N - D_N - t_N^a \\ \lambda_N > t^* - \nu_N T_N - D_N - t_N^a \end{cases} \quad (21)$$

where

$$\overline{K} = \left(\left\lfloor \frac{t^* - \overline{K}_F - \overline{K}_E)}{C_N} \right\rfloor_0 + 1 \right) T_N,$$

$$\overline{K}_F = \sum_{\tau_i \in \Gamma_F(S) \setminus \{\tau_N\}} \overline{dbf}_i(t^* - \Delta_i),$$

$$\overline{K}_E = \sum_{\tau_i \in \Gamma^e(S)} \overline{dbf}_i^e(t^*, \Delta_i, t_i^e).$$

Proof. Following the definition of function $\overline{dbf}_N(t)$, if $t^* < \Delta_N + \nu_N T_N + D_N$ then $\overline{dbf}_N(t^* - \Delta_N)$ corresponds to a value in the piece-wise constant part of the function. Rewriting the latter inequality by expanding $\Delta_N = t_N^a + \lambda_N$, it follows that $\lambda_N > t^* - \nu_N T_N - D_N - t_N^a$. Note that, under such a condition, $\overline{dbf}_N(t^* - \Delta_N) = dbf_N(t^* - \Delta_N)$. Hence, the same approach adopted in the proof of Lemma 5 can be used to derive a lower bound on λ_N. The only difference consists in taking into account the constant terms $\overline{K}_F$ and $\overline{K}_E$, which must be redefined by means of the approximate demand bound functions. Hence, the lemma follows. □

1: **procedure** AADT($\Gamma_F(S)$, $\Gamma^e(S)$, τ_N, x)
2: $\lambda_N \leftarrow 0, \lambda^{(1)} \leftarrow 0, \lambda^{(2)} \leftarrow 0$;
3: $\xi^{oth} \leftarrow \bigcup_{\tau_i \in \{\Gamma_F(S) \setminus \{\tau_N\}\} \cup \Gamma_E(S)} \xi(\tau_i)$;
4:
5: **for** $t \in \xi^{oth} \wedge t \geq t_N^a$ **do**
6: $\lambda' \leftarrow$ LEMMA 8 ($\Gamma_F(S)$, $\Gamma^e(S)$, τ_N);
7: $\lambda'' \leftarrow$ LEMMA 9 ($\Gamma_F(S)$, $\Gamma^e(S)$, τ_N);
8: $\lambda^{(1)} \leftarrow max(\lambda^{(1)}, min(\lambda', \lambda''))$;
9: **end for**
10: **for** $t \in \xi(\tau_N)$ **do**
11: $\lambda' \leftarrow$ LEMMA 6 ($\Gamma_F(S)$, $\Gamma^e(S)$, τ_N);
12: $\lambda^{(2)} \leftarrow max(\lambda^{(2)}, \lambda')$;
13: **end for**
14: $\lambda_N \leftarrow max(\lambda^{(1)}, \lambda^{(2)})$;
15: **for** $i = 1, \ldots, x$ **do** ▷ *Admission delay improvement*
16: $\lambda_N^{ub} \leftarrow \lambda^{(2)}$;
17: $\lambda^{(2)} \leftarrow 0$;
18: **for** $t \in \xi(\tau_N)$ **do**
19: $\lambda' \leftarrow$ LEMMA 7 ($\Gamma_F(S)$, $\Gamma^e(S)$, τ_N, λ_N^{ub}));
20: $\lambda^{(2)} \leftarrow max(\lambda^{(2)}, \lambda')$;
21: **end for**
22: $\lambda_N \leftarrow max(\lambda^{(1)}, \lambda^{(2)})$;
23: **end for**
24: **return** λ_N;
25: **end procedure**

Fig. 5. Pseudo-code for the $AADT$ Approximated Admission Delay for Transients) algorithm.

Lemma 9. *Let $t^* \in \xi(\tau_i)$ be a check-point of Theorem 4 originated by $\tau_i \neq \tau_N$. The check-point is verified if the following system of equations is verified:*

$$\begin{cases} \lambda_N \geq \frac{U_N(t^* - D_N - t_N^a) + \overline{K}_F + \overline{K}_E + C_N - t^*}{U_N} \\ \lambda_N \leq t^* - \nu_N T_N - D_N - t_N^a, \end{cases} \quad (22)$$

where $\overline{K}_F$ and $\overline{K}_E$ are defined in Lemma 8.

Proof. Following the definition of function $\overline{dbf}_N(t)$, if $t^* \geq \Delta_N + \nu_N T_N + D_N$ then $\overline{dbf}_N(t^* - \Delta_N)$ corresponds to a value in the linear part of the function. Rewriting the latter inequality by expanding $\Delta_N = t_N^a + \lambda_N$, it follows that $\lambda_N \leq t^* - \nu_N T_N - D_N - t_N^a$. Under such a condition, $\overline{dbf}_N(t^* - \Delta_N) = C_N + U_N(t^* - \Delta_N - D_N)$; hence, by also expanding Δ_N, Equation (14) becomes

$$C_N + U_N(t^* - t_N^a - \lambda_N - D_N) + \overline{K}_F + \overline{K}_E \leq t^*.$$

The lower bound on λ_N reported in the lemma statement is obtained by solving the latter inequality with respect to λ_N. Hence, the lemma follows. □

The above four lemmas are finally combined into an algorithm for computing an approximate admission delay for τ_N.

Implementation and Complexity. Fig. 5 reports the *AADT* (Approximated Admission Delay for Transients) algorithm. The AADT algorithm starts by initializing $\lambda_N = 0$ (and other auxiliary variables) and by setting ξ^{oth} as the set of check-points that are not originated by τ_N. Then, it proceeds by deriving the lower bounds on λ_N by the check-points in ξ^{oth}, which is accomplished by Lemmas 8 and 9 (lines 6-7). In the reported algorithm, functions Lemma 8 and Lemma 9 return the *minimum* value of λ_N that satisfies the system of

equations in the corresponding lemmas (a value representing infinity is returned when a solution does not exist). By construction of these two lemmas (which match mutually-exclusive scenarios), both the obtained solutions are valid lower-bounds for λ_N: hence, the minimum solution is taken. To obtain a solution that is valid for all the check-points in ξ^{oth}, the maximum admission delay obtained by the lemmas is stored in $\lambda_N^{(1)}$ (line 8s).

Similarly, the subsequent for loop processes all the check-points originated by τ_N (set $\xi(\tau_N)$) by deriving a lower bound for λ_N according to Lemma 6 (line 11). Again, to cope with all such check-points, the maximum admission delay is stored in $\lambda_N^{(2)}$ (line 12). Finally, the algorithm refines the solution by iteratively applying Lemma 7 for x times (as previously discussed below the presentation of the lemma), where $x \in \mathbb{N}_{\geq 0}$ is an input parameter for the algorithm. At every iteration, the solution is updated in the variable λ_N, which is finally returned at the end of the algorithm.

Assuming that each check-point can be processed in constant time (e.g., by storing the sum of demand-bound functions in an incremental fashion whenever a new task joins the system), the AADT algorithm has $\mathcal{O}(|\xi^{oth}| + (x+1)|\xi(\tau_N)|)$ complexity. The cardinality of the involved sets is given by the number of check-points adopted in Theorem 4, hence the algorithm complexity is $\mathcal{O}(\sum_{\tau_i \in \Gamma^*}(\nu_i + 1) + (x+1)(\nu_N + 1))$, with $\Gamma^* = \Gamma_F(S) \cup \Gamma^e(S) \setminus \{\tau_N\}$. The complexity is hence polynomial in the involved parameters.

6 Handling Mode Changes

Although the previous sections were focused on dynamic workloads, the analysis proposed in Section 4 can also be adopted to analyze *static* task sets that exhibit mode changes. To this end, some additional notation and assumptions are needed.

Notation and Assumptions. This section considers a system that can be subject to an arbitrary number n_{mc} of mode changes, where $\mathcal{M}_c = \{mc_1, mc_2, \ldots, mc_{n_{mc}}\}$ denotes the set of all possible mode changes. The h^{th} mode change is defined by a tuple $mc_h = \{M_h^a, M_h^b, t_h^{mc}, \delta_h^{mc}\}$ where t_h^{mc} is the time at which a mode change from mode M_h^a to mode M_h^b is requested, and δ_h^{mc} represents the transition delay, meaning that the tasks of mode M_h^b are allowed to actually join the system only at time $t_h^{mc} + \delta_h^{mc}$. The tasks in the leaving mode M_h^a are allowed to complete the last job released before or at time t_h^{mc}. Each couple of modes M_h^a and M_h^b is associated with a corresponding couple of task sets Γ_h^a and Γ_h^b. Likewise most of the other works that targeted mode changes (e.g., [14] and [16]), this section assumes that (i) the transition delay δ_h^{mc} of each mode change mc_h is assumed to be fixed, (ii) the task sets Γ_h^a and Γ_h^b are schedulable in steady-state conditions, and (iii) scheduling transients do not overlap (i.e., at most one mode change per busy period). Please refer to Appendix A.3, available online, for a discussion on how (iii) can be relaxed.

Analysis. Now, it is possible to proceed by showing how mode changes can be mapped into the model based on the sequence of events introduced in Section 3. For a given mode change mc_h, the idea is to consider the first task set Γ_h^a as the one that is present in the system before the beginning of a sequence S that should model the mode change from Γ_h^a to Γ_h^b. The results of Theorem 2 can then be adopted to analyze mc_h by constructing the sequence S as the concatenation of two sub-sequences: (i) S^E, which comprises the exit events of all tasks that *leave* the system, i.e., those in the set $\Gamma_h^a \setminus \Gamma_h^b$, and (ii) S^A, which comprises the arrival events of all tasks that *join* the system, i.e., those in the set $\Gamma_h^b \setminus \Gamma_h^a$, thus obtaining:

- $S^E = \{\cup_{\tau_i \in \{\Gamma_h^a \setminus \Gamma_h^b\}} < \tau_i, t_h^{mc}, \texttt{EXIT} >\}$,
- $S^A = \{\cup_{\tau_i \in \{\Gamma_h^b \setminus \Gamma_h^a\}} < \tau_i, t_h^{mc} + \delta_h^{mc}, \texttt{ARRIVAL} >\}$,
- $S = \{S^E, S^A\}$.

Analogously, the task sets introduced in Section 3 can be defined as follows:

- $\Gamma^O = \Gamma_h^b \cap \Gamma_h^a$;
- $\Gamma^e(S) = \Gamma_h^a \setminus \Gamma^O$;
- $\Gamma^a(S) = \Gamma_h^b \setminus \Gamma^O$.

Finally, to ensure a consistent matching between the two models, the parameters t_i^e and λ_i are defined as follows:

- $\forall \tau_i \in \Gamma^e(S), t_i^e = t_h^{mc}$, $t_i^a = 0$, and $\lambda_i = 0$;
- $\forall \tau_i \in \Gamma^a(S), t_i^a = t_h^{mc}$ and $\lambda_i = \delta_h^{mc}$.

Although this model transformation allows adopting the results presented in the previous sections, a major issue is still present. When performing an off-line analysis of static task sets with mode changes, the times t_h^{mc} at which mode changes occur may not be known a priori, i.e., they may be triggered at different times when the system is running. A safe analysis must therefore cope with all valid values for times t_h^{mc}.

Theoretically speaking, if Theorem 2 is verified $\forall t_h^{mc} \geq 0$, then the system subject to mode change mc_h is schedulable. However, this approach does not allow realizing a practical schedulability test, as the continuous domain of t_h^{mc} would have to be explored. Nevertheless, by studying the equations involved in Theorem 2, it is possible to devise a safe domain discretization for t_h^{mc}, as it is expressed by the following lemma.

Lemma 10. *Let $\widehat{dbf}_i^e(t, \Delta_i, t_h^{mc})$ be a simple upper bound of function $dbf_i^e(t, \Delta_i, t_h^{mc})$ (defined in Equation (5)) where $c_i = C_i$. If Theorem 2 holds by replacing $dbf_i^e(t, \Delta_i, t_h^{mc})$ with $\widehat{dbf}_i^e(t, \Delta_i, t_h^{mc})$*

$$\forall t_h^{mc} \in \bigcup_{\tau_i \in \Gamma^e(S)} \{fT_i : f \in \mathbb{N}_{\geq 0}\}, \tag{23}$$

then Theorem 2 also holds $\forall t_h^{mc} \geq 0$.

Proof. The lemma follows by studying the dependency of Equation (8) on t_h^{mc} after replacing $dbf_i^e(t, \Delta_i, t_h^{mc})$ with $\widehat{dbf}_i^e(t, \Delta_i, t_h^{mc})$. First note that function $\widehat{dbf}_i^e(t, \Delta_i, t_h^{mc})$ depends on t_h^{mc} by means of terms $t_h^{mc} = t_i^e$, which in turn affect the definition of α_i in Equation (6). Since for all tasks $\tau_i \in \Gamma^e(S)$, $\Delta_i = 0$, then α_i, and consequently $\widehat{dbf}_i^e 1(t, \Delta_i, t_h^{mc})$, changes only for $t_h^{mc} = fT_i$ with $f \in \mathbb{N}_{\geq 0}$.

Second, observe that function $dbf_i(t - \Delta_i)$ depends on t_h^{mc} by means of $\Delta_i = t_h^{mc} + \delta_h^{mc}$ and that such a function is non-increasing in Δ_i. Hence, fixed a time t and for a given interval $[fT_j, (f+1)T_j)$, $dbf_i(t - \Delta_i)$ is maximal in $t_h^{mc} = fT_j$. Consequently, if Equation (8) holds in

```
1: procedure MCA(M_c)
2:     for mc_h ∈ M_c do
3:         Γ^O ← Γ^b_h ∩ Γ^a_h;
4:         Γ^e(S) ← Γ^a_h \ Γ^O;
5:         Γ^a(S) ← Γ^b_h \ Γ^O;
6:         Γ_F(S) = Γ_O ∪ {Γ^a(S) \ Γ^e(S)};
7:         sched ← LEMMA 10 (Γ_F(S), Γ^e(S));
8:         if sched is FALSE then
9:             return FALSE;
10:        end if
11:    end for
12:    return TRUE;
13: end procedure
```

Fig. 6. Pseudo-code for the MCA (Mode-Change Analysis) algorithm.

$t_h^{mc} = fT_j$, then it cannot be violated for $t_h^{mc} \in (fT_j, (f+1)T_j)$. Finally, note that the term $t^*(\tau_i)$ in Theorem 2 also changes for $t_h^{mc} = fT_i$ with $f \in \mathbb{N}_{\geq 0}$. Hence, the lemma follows. □

Since all scheduling transients are exhausted at the first idle-time, the mode-change times t_h^{mc} considered in the previous lemma can be limited to the length of longest busy-period [37] of Γ_h^a.

Fig. 6 reports the MCA (Mode-Change Analysis) algorithm, which exploits Lemma 10 to implement a schedulability test.

7 Experimental Results

This section presents the results of two experimental studies that have been conducted to evaluate the proposed approaches. The first one is aimed at assessing the performance of the ADT and AADT algorithms proposed in Section 5, comparing the admission delays provided by the two algorithms in the context of dynamic real-time workloads. The second experimental study is aimed at evaluating the schedulability analysis for task sets with mode changes proposed in Section 6 (algorithm MCA), which is compared against the SUBI algorithm proposed by Fisher and Ahmed [14] and against the schedulability analysis proposed by Lee and Shin in [16].

7.1 Dynamic Workload: ADT versus AADT

This experimental study has been performed by simulating the protocol presented in Section 5. The protocol has been stimulated by synthetic sequences of events, which represent the arrival and the exit of tasks at randomly-generated times. The experimentation aims at computing the average of the admission delay bounds (computed with the algorithms presented in the previous section) incurred by tasks during such sequences. To ensure a meaningful comparison, the measured admission delays are normalized to the task periods, i.e., for an arbitrary task τ_i, the parameter λ_i/T_i is measured.

Sequence Generation. Let U_F be a configurable generation parameter that represents the steady-state utilization of the system. First, a task set Γ is generated, which contains the tasks that are present in the system at the beginning of the sequence. These tasks are considered to be simultaneously released at $t = 0$, where the busy-period under analysis starts. Given a number of tasks $n = |\Gamma|$ and a target utilization $U_F/2$, the utilization of each task $\tau_i \in \Gamma$ has been generated using the UUnifast algorithm [40]. Then, a sequence S of N events was generated by alternating exit and arrival events. When generating an arrival event, the utilization of the corresponding task τ_i was generated to keep the total utilization U_F constant. To enable a meaningful comparison between the ADT and AADT algorithms, the generation discards the tasks that do not lead to a schedulable system in steady-state according to Theorem 4, where the parameter ν_i has been set to $\nu_i = 2$ for all tasks. Exit events were generated by selecting a random task as the one that leaves the system: to avoid inconsistencies in the comparison, only the tasks in Γ are selected, otherwise the two algorithms would be subject to different sequence of events.[2] For all tasks, periods were generated in the interval $[1, 1000]$ ms with uniform distribution, and the WCETs were obtained as $C_i = U_i T_i$. Relative deadlines have been generated with uniform distribution in the interval $[C_i + \beta(T_i - C_i), T_i]$, where $\beta \in [0, 1]$ is another generation parameter. The times t_k at which the exit events occur have been proportionally generated with respect to the tasks' periods, i.e., $t_k = xT_{\mathrm{AVG}}$, where T_{AVG} is the average period of tasks for which an exit event has not yet been generated, and x is a real number that was randomly generated with uniform distribution in $[0, 1]$. Similarly, the times t_k in which arrival events occur have been generated as $t_k = \sigma x T_{\mathrm{AVG}}$, where the additional parameter σ was introduced to better regulate the distance from the last generated event. The generation of a sequence terminates when all the tasks in the set Γ are exited.

Admission Delays. The algorithms ADT and AADT have been compared with a multidimensional exploration of the parameters that control the workload generation. In particular, the generation parameters have been varied as follows:

- $U_F \in [0.5, 0.95]$, with step 0.5;[3]
- n from 2 to 20, with step 1;
- β in the set $\{0.1, 0.3, 0.6, 0.9\}$; and
- σ in the set $\{0.001, 0.05, 0.15\}$.

For each configuration of the parameters, 10000 sequences of events have been generated. Each sequence has been then processed with both the ADT and AADT algorithms averaging the normalized admission delays incurred by tasks. Different variants of the AADT algorithm have been compared, considering 0, 1, 3, 5, 10 and 15 iterations for Lemma 7. In the following, the notation AADT(k) is used to denote that the AADT algorithm is adopted with k iterations for Lemma 7. When processing a sequence with a given algorithm, the admission delays are not computed whenever a task arrival event occurs after the length of the corresponding longest busy-period: such events are also assigned with an admission delay equal to zero.

2. Note that, at a given time t, a task that requested to join the system at a time $t_1 < t$ may not be actually admitted (i.e., when $t_1 + \Delta_i > t$) and hence it may not be eligible for leaving the system. This depends on the admission delays and then on the adopted algorithm.

3. A preliminary experimental study showed that for $U_F < 0.5$ the admission delays are very low (close to zero) with both the proposed algorithms.

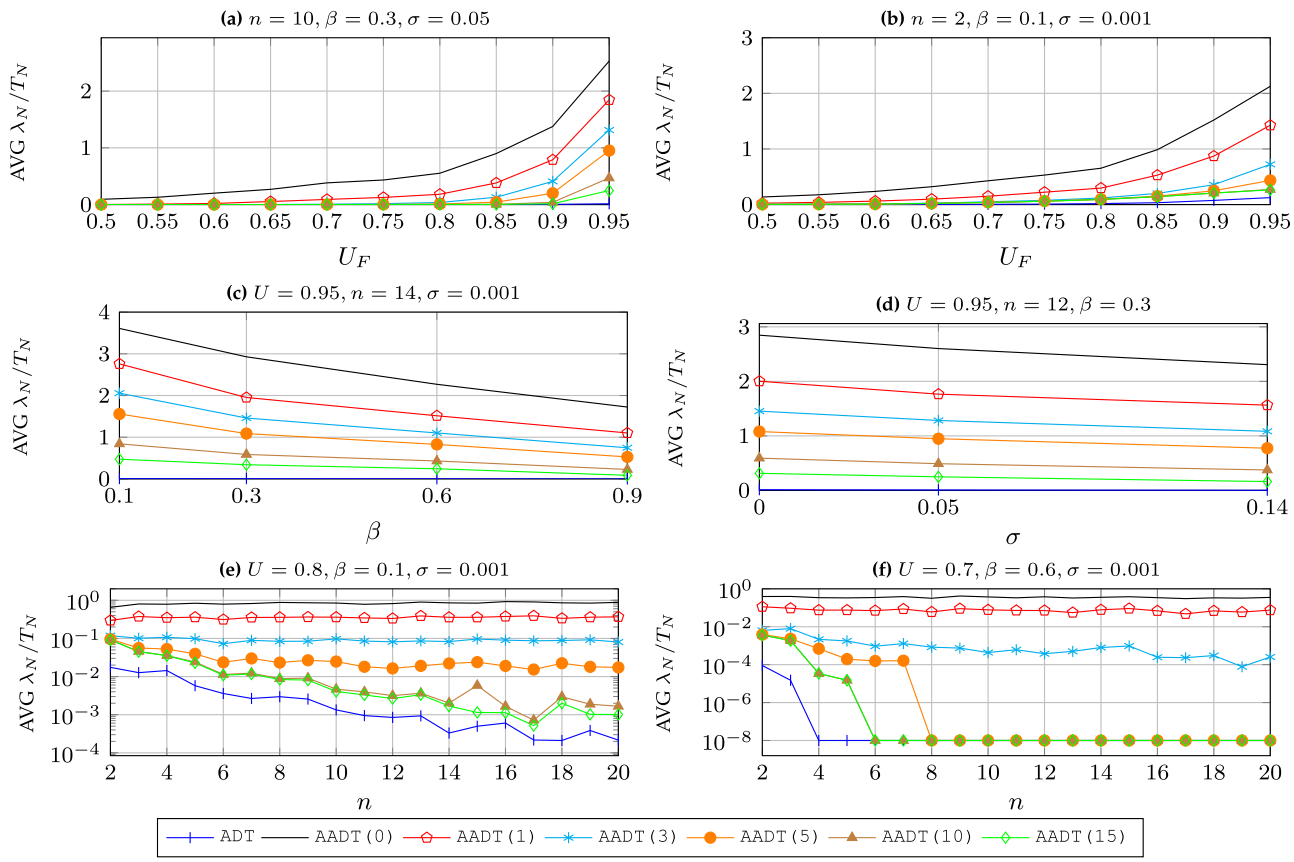

Fig. 7. Average normalized admission delay (with respect to the period of each admitted task) obtained by using the ADT and AADT(k) algorithms as a function of the number of tasks (insets (e) and (f)), the utilization U_F (insets (a) and (b)), and the parameters σ and β (insets (c) and (d), respectively).

The experimental results for six representative configurations are reported in Fig. 7, where the dependency of the admission delay on various parameters can be observed.

At a high level, the ADT algorithm always provides very limited admission delays. The same holds for the AADT(k) when $k \geq 10$. For instance, this can be observed as a function of the utilization U_F from Figs. 7a and 7b, where different values for n, β and σ have been adopted.

Fig. 7c illustrates the average admission delay as a function of the parameter β, which has been used to control the generation of relative deadlines. From the plot, it emerges that the tighter the deadline, the higher the admission delays obtained with all the algorithms. A similar trend can be observed in Fig. 7d, where the distance in time between arrival and exit events has been varied by controlling the parameter σ. Finally, Figs. 7e and 7f show that, when adopting a low number of iterations k, the admission delays obtained with the AADT(k) remain almost constant as the number of tasks increases. Conversely, for the more precise algorithms, the admission delays decrease as the number of tasks increases, also showing a saturation to extremely low delays at some cut-off values of n. Note that these two figures have a logarithmic scale.

Additional Experiments. Additional experimental results have been carried out to further evaluate the proposed approach. To this aim, the ADT algorithm has been tested by using Theorem 1 as admission test, which is possible as it has not been compared against AADT. Albeit the adoption of Theorem 1 in place of Theorem 3 allows testing the algorithm with a wider range of task sets, Fig. 8a shows that the normalized admission delay is still very low. Finally, Fig. 8b targets the case of task sets composed of only implicit-deadline tasks, and compares the proposed approaches with the methodology for computing the admission delay proposed in [3], which is denoted in the figure as Buttazzo*. The method proposed in [3] requires the knowledge of the remaining execution time of the last job of each task $\tau_i \in \Gamma^e(S)$, which is not available in our sequence-based model. To enable a comparison, the remaining execution times have been considered to be equal to the corresponding WCETs.

Running Times. The maximum running times of the tested algorithms have been measured during the experimental study discussed above. The experiments have been performed on a machine equipped with an Intel Core i7-6700K @ 4.00GHz. Algorithms have been realized with literal implementations (i.e., not designed for being extremely efficient). The Microsoft VC++2015 compiler has been used and running times have been measured by means of the Windows API.[4] Fig. 9 shows that ADT requires a considerable amount of time (up to 43 ms), thus resulting unsuitable for online admission control. Conversely, running-times observed with AADT(15) are always under 500 μs.

4. Wall-clock has been measured by executing the experiments on a dedicated processor. Therefore, the measurements also include some additional overhead (e.g., execution of the kernel). A preliminary experiment excluded the possibility of using the Windows API aimed at only measuring the time spent in the process (e.g., *GetProcessTimes()*), as the running time of the AADT algorithm is often under (or comparable with) the precision offered by that API.

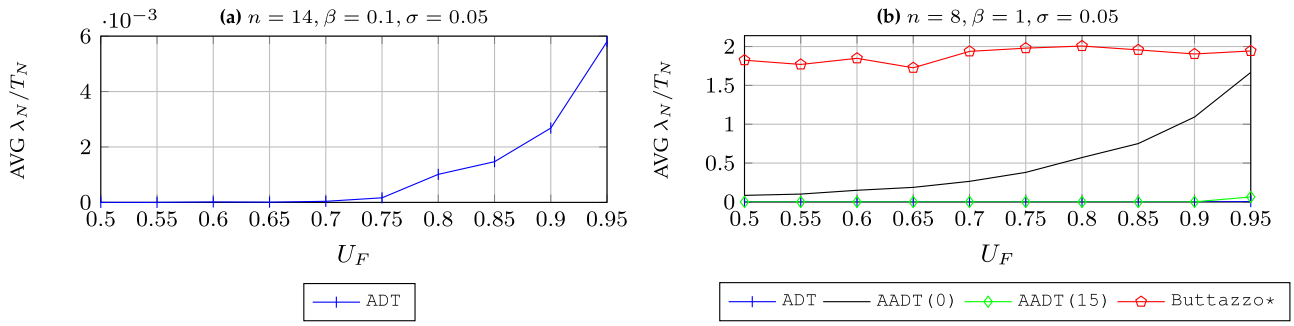

Fig. 8. Average normalized admission delay (with respect to the period of each admitted task) obtained by using the ADT and AADT(k) algorithms as a function of the utilization U_F, and the parameters σ and β. Inset (a) reports only the performance of the ADT algorithm, and hence uses Theorem 1 as admission test. Inset (b) compares the proposed approaches with [3] for implicit-deadline tasks.

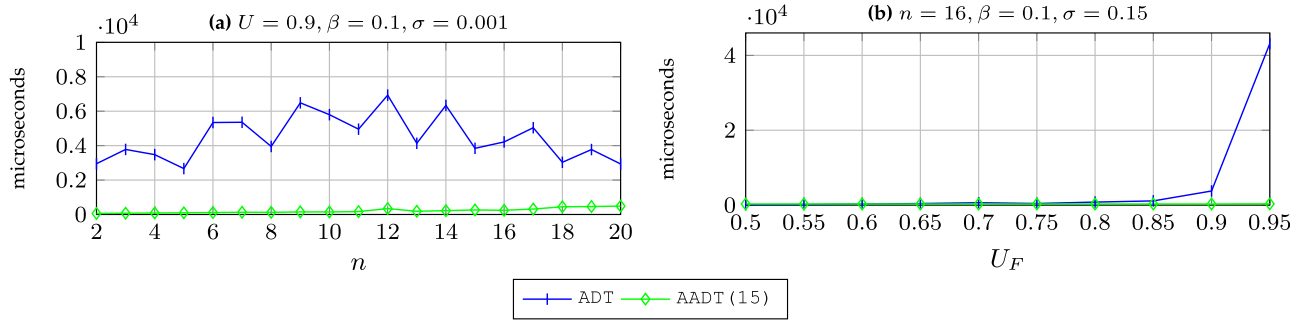

Fig. 9. Maximum observed running times (in microseconds) obtained by running ADT and AADT(15) with respect to the number of tasks (inset (a)) and the utilization U_F (inset (b)).

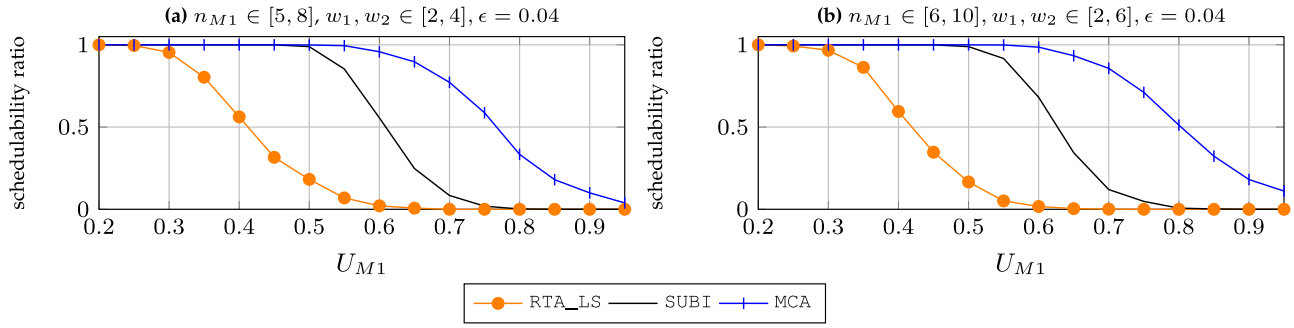

Fig. 10. Schedulability ratio of different schedulability analyses for mode changes with parameters uniformly distributed in: (a) $n_{M1} \in [5, 8]$ and $w_1, w_2 \in [2, 4]$, (b) $n_{M1} \in [6, 10]$ and $w_1, w_2 \in [2, 6]$).

7.2 Mode-Change

A second experimental study has been carried out to evaluate the proposed analysis for static task sets with mode changes (MCA algorithm of Section 6), by comparing it with SUBI algorithm proposed by Fisher et al. in [14] and the schedulability analysis proposed by Lee and Shin [16] (referred to as RTA_LS in the following). A MATLAB implementation of the SUBI algorithm has been kindly provided by the authors.

This experimental study measured the schedulability ratio, i.e., the ratio between the number of schedulable task sets (in the presence of mode changes) and the overall number of generated task sets.

Workload Generation. The evaluation considered task sets with two modes M_a and M_b (bi-directional mode changes are possible). Given a target utilization U_{M1} of mode M_1, the corresponding task set Γ_{M1} was generated with UUnifast by fixing a number of task n_{M1}. The second task set Γ_{M2} was obtained by the first one by removing w_1 randomly-selected tasks and then adding w_2 new tasks. The latter were generated with UUnifast, setting a target utilization U_{M2} that is randomly generated with uniform distribution in the range $[U_{M1} - \epsilon, U_{M1} + \epsilon]$, where ϵ is a generation parameter. A preliminary experimental study conducted on the implementation of SUBI showed that the running time for a simple system composed of 4 tasks and 2 operating modes (specifically, the one depicted in Fig. 1 with the parameters scaled by 10^3) is about 6 minutes. It is worth observing that this large running time may depend on the fact that, differently from MCA, SUBI is capable of handling overlapping mode changes. Nevertheless, it has been found that the running time is largely dependent on the tasks' periods (likely because the analysis conditions are checked for all integers within a time window), which prevented us to perform experiments with random periods within a given range. To cope with this issue, the periods have been randomly chosen from the bucket $\{10, 20, 25, 30, 40, 45, 50, 60, 65, 80\}$ ms. Relative deadlines were computed as in Section 7.1 with $\beta = 0.5$. The task sets that resulted not schedulable in steady-state conditions according to Theorem 1 have been discarded.

Experiments. Note that the RTA_LS algorithm only supports mode changes without admission delays. To allow a fair comparison, the three algorithms have been configured to work under the same conditions by setting $\delta_{mc}^{h} = 0$.

The schedulability performance of the three algorithms has been studied as a function of the utilization U_{M_1}. Fig. 10 reports the experimental results under two representative configurations. As it emerges from the graphs, the proposed algorithm (MCA) significantly outperforms both SUBI and RTA_LS, especially for significant utilization values ($U_{M_1} > 0.6$). In particular, note that MCA reaches a performance improvement up to 96 percent with respect to RTA_LS ($U_{M1} = 0.6$) and up to 72 percent with respect to SUBI ($U_{M1} = 0.7$).

8 Conclusions

This paper presented a schedulability analysis framework for real-time dynamic workloads that can experience scheduling transients under EDF scheduling. The framework is able to deal with constrained-deadline tasks and is entirely built upon a new modeling approach based on a sequence of events. Leveraging this proposal, an on-line protocol has been presented to handle the admission control of dynamic workloads , which comes with algorithms for computing the admission delays that new tasks must wait before joining the system. Both pseudo-polynomial-time and polynomial-time solutions have been proposed. Finally, the paper also showed how the proposed analysis can be used off-line for guaranteeing the schedulability of static task sets with mode changes. Experimental results showed that the polynomial-time algorithm allows reaching an empirical performance that is close to the pseudo-polynomial-time solution, while incurring a very limited running time. Furthermore, the proposed approach has been also shown to be effective in analyzing static task sets with mode changes, exhibiting a performance improvement in terms of schedulability ratio up to 72 percent over state-of-the-art methods.

Interesting research can follow from this work, including the derivation of methods for computing admission delays that are tailored for semi-partitioned scheduling, the study of analysis techniques to handle overlapping mode changes in the presence of static task sets, and the design of further approximation schemes for the proposed approach.

Acknowledgments

The authors would like to thank Masud Ahmed and Nathan Fisher for providing the code of the SUBI algorithm [14].

References

[1] T. Cucinotta, L. Abeni, L. Palopoli, and G. Lipari, "A robust mechanism for adaptive scheduling of multimedia applications," *ACM Trans. Embedded Comput. Syst.*, vol. 10, no. 4, Nov. 2011, Art. no. 124.

[2] K. Konstanteli, T. Cucinotta, K. Psychas, and T. Varvarigou, "Admission control for elastic cloud services," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 24–29, 2012, pp. 41–48.

[3] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management," *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289–302, Mar. 2002.

[4] K. W. Tindell, A. Burns, and A. J. Wellings, "Mode changes in priority preemptively scheduled systems," in *Proc. Real-Time Syst. Symp.*, Dec. 2–4, 1992, pp. 100–109.

[5] B. Brandenburg and M. Gul, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *Proc. 37th IEEE Real-Time Syst. Symp.*, Nov. 29 - Dec. 2, 2016, pp. 99–110.

[6] D. Casini, A. Biondi, and G. Buttazzo, "Semi-partitioned scheduling of dynamic real-time workload: A practical approach based on analysis-driven load balancing," in *Proc. 29th Euromicro Conf. Real-Time Syst.*, Jun. 27–30, 2017, pp. 13:1–13:23.

[7] G. Fohler, "Realizing changes of operational modes with a pre run-time scheduled hard real-time system," *Responsive Comput. Syst.*, vol. 7, pp. 287–300, Mar. 1993.

[8] Q. Guangming, "An earlier time for inserting and/or accelerating tasks," *Real-Time Syst.*, vol. 41, pp. 181–194, 2009.

[9] P. Rattanatamrong and J. A. B. Fortes, "Mode transition for online scheduling of adaptive real-time systems on multiprocessors," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 28–31, 2011, vol. 1, pp. 25–32.

[10] B. Andersson, "Uniprocessor EDF scheduling with mode change," in *Proc. 12th Int. Conf. Principles Distrib. Syst.*, Dec. 15–18, 2008, pp. 572–577.

[11] V. Nelis, J. Goossens, and B. Andersson, "Two protocols for scheduling multi-mode real-time systems upon identical multiprocessor platforms," in *Proc. 21st Euromicro Conf. Real-Time Syst.*, Jul. 1–3, 2009, pp. 151–160.

[12] V. Nelis, J. Marinho, B. Andersson, and S. M. Petters, "Global-EDF scheduling of multimode real-time systems considering mode independent tasks," in *Proc. 23rd Euromicro Conf. Real-Time Syst.*, Jul. 6–8, 2011, pp. 205–214.

[13] L. Phan, I. Lee, and O. Sokolsky, "Compositional analysis of multimode systems," in *Proc. 22st Euromicro Conf. Real-Time Syst.*, Jul. 6–9, 2010, pp. 197–206.

[14] N. Fisher and M. Ahmed, "Tractable real-time schedulability analysis for mode changes under temporal isolation," in *Proc. 9th IEEE Symp. Embedded Syst. Real-Time Multimedia*, Oct. 13–14, 2011, pp. 130–139.

[15] N. Stoimenov, S. Perathoner, and L. Thiele, "Reliable mode changes in real-time systems with fixed priority or EDF scheduling," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, Apr. 20–24, 2009, pp. 99–104.

[16] J. Lee and K. Shin, "Schedulability analysis for a mode transition in real-time multi-core systems," in *Proc. IEEE 34th Real-Time Syst. Symp.*, Dec. 2013, pp. 11–20.

[17] A. Block and J. H. Anderson, "Accuracy versus migration overhead in real-time multiprocessor reweighting algorithms," in *Proc. 12th Int. Conf. Parallel Distrib. Syst.*, Jul. 12–15, 2006, pp. 355–364.

[18] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-Time Syst.*, vol. 26, no. 2, pp. 161–197, Mar. 2004.

[19] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," *Real-Time Syst.*, vol. 1, pp. 243–264, 1989.

[20] M. Ahmed, N. Fisher, and D. Grosu, "A parallel algorithm for EDF-Schedulability analysis of multi-modal real-time systems," in *Proc. IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 19–22, 2012, pp. 154–163.

[21] N. Stoimenov, L. Thiele, L. Santinelli, and G. Buttazzo, "Resource adaptations with servers for hard real-time systems," in *Proc. 10th ACM Int. Conf. Embedded Softw.*, 2010, pp. 269–278.

[22] L. Santinelli, G. Buttazzo, and E. Bini, "Multi-moded resource reservations," in *Proc. 17th IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 11–13, 2011, pp. 37–46.

[23] P. Rattanatamrong and J. A. B. Fortes, "Mode transition for online scheduling of adaptive real-time systems on multiprocessors," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 28–31, 2011, pp. 25–32.

[24] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 4, pp. 553–566, Apr. 2009.

[25] H. Kopetz, R. Nossal, R. Hexel, A. Krger, D. Millinger, R. Pallierer, C. Temple, and M. Krug, "Mode handling in the time-triggered architecture," *Control Eng. Practice*, vol. 6, no. 1, pp. 61–66, 1998.

[26] F. Heilmann, A. Syed, and G. Fohler, "Mode-changes in COTS time-triggered network hardware without online reconfiguration," *SIGBED Rev.*, vol. 13, no. 4, pp. 55–60, 2016.

[27] A. Block, J. H. Anderson, and G. Bishop, "Fine-grained task reweighting on multiprocessors," in *Proc. 11th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Jul. 17–19, 2005, pp. 429–435.
[28] B. Andersson and C. Ekelin, "Exact admission-control for integrated aperiodic and periodic tasks," in *Proc. 11th IEEE Real Time Embedded Technol. Appl. Symp.*, Mar. 7–10, 2005, pp. 76–85.
[29] A. Masrur, D. Mller, and M. Werner, "Bi-level deadline scaling for admission control in mixed-criticality systems," in *Proc. IEEE 21st Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 19–21, 2015, pp. 100–109.
[30] S. Baruah, V. Bonifaci, G. D'angelo, H. Li, A. Marchetti-Spaccamela, S. Van Der Ster, and L. Stougie, "Preemptive uniprocessor scheduling of mixed-criticality sporadic task systems," *J. ACM*, vol. 62, no. 2, 2015, Art. no. 14.
[31] X. Gu and A. Easwaran, "Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems," *ACM Trans. Embed. Comput. Syst.*, vol. 17, no. 1, 2018, Art. no. 24.
[32] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Syst.*, vol. 2, no. 4, pp. 301–324, 1990.
[33] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Softw.: Practice Experience*, vol. 46, no. 6, pp. 821–839, 2016.
[34] "AUTOSAR 4.2 OS specification," 2015. [Online]. Available: http://www.autosar.org/
[35] K. Tindell and A. Alonso, "A very simple protocol for mode changes in priority preemptive systems," Universidad Politecnica de Madrid, Madrid, Spain, 1996.
[36] G. C. Buttazzo, *Hard Real-Time Comput. Syst.: Predictable Scheduling Algorithms and Appl., Third Edition*. Springer, 2011.
[37] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.
[38] M. Spuri, "Analysis of deadline scheduled real-time systems," INRIA, France, Rep. no. RR-2772, 1996.
[39] N. Fisher, T. P. Baker, and S. Baruah, "Algorithms for determining the demand-based load of a sporadic task system," in *Proc. 12th IEEE Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, Aug. 16–18, 2006, pp. 135–146.
[40] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, pp. 129–154 , 2005.

Daniel Casini received the graduate (cum laude) degree in embedded computing systems engineering, and the master's degree jointly offered by the Scuola Superiore Sant'Anna of Pisa and University of Pisa. He is a PhD fellow at the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna of Pisa, working under the supervision of Prof. Alessandro Biondi and Prof. Giorgio Buttazzo. His research interests include software predictability in multi-processor systems, schedulability analysis, synchronization protocols, and the design and implementation of real-time operating systems and hypervisors.

Alessandro Biondi received the graduate (cum laude) degree in computer engineering from the University of Pisa, Italy, within the excellence program, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna under the supervision of Prof. Giorgio Buttazzo and Prof. Marco Di Natale. He is an not a assistant professor with the Real-Time Systems (ReTiS) Laboratory of the Scuola Superiore Sant'Anna. In 2016, he has been visiting scholar at the Max Planck Institute for Software Systems, Germany. His research interests include design and implementation of real-time operating systems and hypervisors, schedulability analysis, cyber-physical systems, synchronization protocols, and component-based design for real-time multiprocessor systems. He was recipient of five Best Paper Awards, one Outstanding Paper Award, and the EDAA Dissertation Award 2017.

Giorgio Buttazzo received the graduate degree in electronic engineering from the University of Pisa, in 1985, the MS degree in computer science from the University of Pennsylvania, in 1987, and the PhD degree in computer engineering from the Scuola Superiore Sant'Anna of Pisa, in 1991. He is full professor of computer engineering at the Scuola Superiore Sant'Anna of Pisa. He is editor-in-chief of Real-Time Systems, associate editor of the ACM Transactions on Cyber-Physical Systems, and IEEE fellow since 2012. He has authored 7 books on real-time systems and more than 200 papers in the field of real-time systems, robotics, and neural networks.