Polynomial-Time Exact Schedulability Tests for Harmonic Real-Time Tasks

Vincenzo Bonifaci*, Alberto Marchetti-Spaccamela†, Nicole Megow‡, Andreas Wiese§
*Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti" (IASI-CNR), Rome, Italy
Email: vincenzo.bonifaci@iasi.cnr.it
†Sapienza University of Rome, Italy. Email: alberto.marchetti@dis.uniroma1.it
‡Technische Universität Berlin, Germany. Email: nmegow@math.tu-berlin.de
§Max-Planck-Institut für Informatik, Saarbrücken, Germany. Email: awiese@mpi-inf.mpg.de

***Abstract*—We study the preemptive scheduling of real-time sporadic tasks on a uniprocessor. We consider both fixed priority (FP) scheduling as well as dynamic priority scheduling by the Earliest Deadline First (EDF) algorithm. We investigate the problems of testing schedulability and computing the response time of tasks. Generally these problems are known to be computationally intractable for task systems with constrained deadlines. In this paper, we focus on the particular case of task systems with *harmonic* period lengths, meaning that the periods of the tasks pairwise divide each other. This is a special case of practical relevance.**

We present provably efficient exact algorithms for constrained-deadline task systems with harmonic periods. In particular, we provide an exact polynomial-time algorithm for computing the response time of a task in a system with an arbitrary fixed priority order. This also implies an exact FP-schedulability test. For dynamic priority scheduling, we show how to test EDF-schedulability in polynomial time. Additionally, we give a very simple EDF-schedulability test for the simpler case where relative deadlines and periods are jointly harmonic.

I. Introduction

The *sporadic task model* [29] is a widely used formal model for representing recurrent real-time task systems. We consider the preemptive uniprocessor scheduling of sporadic tasks with *harmonic* period lengths, which means that the periods of the tasks divide each other. Harmonic periods occur in a number of applications domains [12], [30], [34] and there is experimental evidence that they may allow a larger processor utilization [9], [16], [23]. In this paper, we formally prove that constrained-deadline[1] task systems with harmonic periods admit polynomial-time exact schedulability tests.

Schedulability analysis is used to a priori assess the behavior of a real-time system. Given a set of real-time tasks, schedulability analysis determines whether this set can be guaranteed to always meet the deadlines at runtime. A schedulability test is *sufficient* if all task sets deemed to be schedulable by the test are in fact schedulable. Similarly, a schedulability test is *necessary* if all task sets deemed to be unschedulable by the test are in fact unschedulable. Schedulability tests that are both sufficient and necessary are referred to as *exact*.

[1] A *constrained-deadline* task set is such that the relative deadline of each task does not exceed its period length. When each relative deadline is equal to the corresponding period length, the task set is called *implicit-deadline*.

In this paper, we consider both *fixed priority* and *dynamic* scheduling. In fixed priority (FP) scheduling an arbitrary priority ordering of all tasks is specified. Jobs are scheduled preemptively according to this order. That is, at the arrival of a job of a higher priority task, any executing job of a lower priority task is preempted. Among all possible priority orders, some well-studied ones are Rate Monotonic (RM), where tasks are ordered non-decreasingly by their period lengths, and Deadline Monotonic (DM), where tasks are ordered non-decreasingly by their relative deadlines.

A well-known method to perform schedulability analysis for fixed priority scheduling is to compute the worst-case response time of each task. This is known as Response-Time Analysis (RTA) [2], [22], [37]. The *response time* of a task is the maximum amount of time that may elapse between the arrival of a job of the task and its completion. Known methods for computing the response time are iterative and not known to be polynomial-time in the input size. They are typically unsuitable for large instances [22]. Indeed, it is known that computing the response time for RM-scheduling is NP-hard [13]. Therefore, it is unlikely that exact polynomial-time RTA-based schedulability tests exist.

In the case of dynamic scheduling we consider the Earliest Deadline First (EDF) algorithm, which schedules at any time an available job whose absolute deadline is closest, possibly preempting a running job with later deadline. EDF is an optimal scheduling algorithm for a set of preemptive jobs on a single processor [11], [28], meaning that it constructs a feasible schedule (a schedule in which all jobs meet their deadlines) whenever a feasible schedule exists. A well-known method to perform EDF-schedulability analysis is based on the *demand bound function* [6]: roughly speaking, the total processing demand is computed for certain time intervals and compared with the available processing capacity in those intervals. The running time of the test depends on the number of time intervals to be considered; no polynomial-time bound on the complexity of such an approach is known. Indeed, the problem of testing EDF-schedulability has been shown to be coNP-hard even in the case of constrained-deadline task sets [15]; therefore, it is unlikely that a polynomial-time exact EDF-schedulability test exists.

It is noteworthy that the two above-mentioned intractability

results (NP-hardness of response-time computation [13] and coNP-hardness of testing EDF-schedulability [15]) heavily rely on the fact that the period lengths of an instance can have a complicated algebraic structure. Therefore, for solving the two problems above exactly, one has to cope with complex subproblems from the domain of algorithmic number theory. In fact, the two latter results hinge on the NP-hard *Directed Diophantine Approximation* problem [14]. However, in practical applications the structure of the arising period lengths might be much simpler than in a possible theoretical worst-case scenario.

Therefore, it is of interest to determine under which conditions polynomial-time exact schedulability tests are possible. Despite all research effort, only for very special cases a positive answer is known. A remarkable result has been obtained by Liu and Layland in 1973 [28], who showed that in the case of implicit-deadline task systems an exact schedulability test for EDF systems can be easily performed by verifying whether the total utilization of the task set is less than or equal to 1.

The class of task systems with harmonic periods is appealing from a practical point of view, but its power is not well understood from a theoretical point of view. In the specific case of n implicit-deadline tasks with harmonic periods, it is known[2] that a sufficient condition for RM-schedulability is that the total utilization of the task set is at most 1; whereas for non-harmonic periods, only a utilization bound of $n(2^{1/n}-1)$ (which approaches $\ln 2 \approx 0.693$, for large n) yields a sufficient condition [28]. Research effort has been dedicated to extend the result to other system models. For example, in [23] the utilization bounds of [28] have been generalized using the notion of harmonic chains. In [35] the authors propose to transform the periods of an implicit-deadline task set to make them nearly harmonic. However, in the case of harmonic, constrained-deadline task sets, the complexity of known exact schedulability tests is only pseudopolynomial.

A. This paper

We study preemptive real-time schedulability tests for sporadic task sets with constrained deadlines on a uniprocessor, under the assumption that the period lengths of the tasks are harmonic, that is, they pairwise divide each other. As we already observed, harmonic periods are relevant in a number of applications, such as avionics (see, for example, [30]). We show that such task sets admit exact polynomial time tests, thus bypassing the previously discussed computational hardness results.

We first consider fixed priority scheduling. We present an exact polynomial-time algorithm which computes the response time of a task of a given task set for an arbitrary priority ordering (Section III). A key difficulty when computing the response time is that the inequality for upper bounding the response time (see Inequality (2)) might be satisfied only at some isolated points in time, and such points may be hard to find. However, in the case of harmonic period lengths we prove that if the inequality holds at time point t, then it also holds at any time after t that is a multiple of some period length p_j (see Figure 1 and Lemma 2). This allows us to find the response time with a binary search type procedure having an overall complexity $\mathcal{O}(n \cdot \log P)$, where P is the largest period length. As a corollary, we obtain an exact polynomial-time test that decides whether a set of tasks with harmonic period lengths is FP-schedulable under a given ordering.

We then consider dynamic scheduling algorithms. Our main result is a polynomial-time algorithm that tests whether the given set of tasks is feasible on one processor (Section IV-B). It is known that testing uniprocessor feasibility is equivalent to testing whether there is a feasible schedule for the synchronous arrival sequence of jobs [4]. Intuitively, one might want to simulate a suitable scheduling algorithm (such as EDF) in order to verify whether a schedule exists for this job sequence. However, this approach would be inefficient. Instead, we construct a schedule, specially designed for the synchronous arrival sequence, that can be described compactly. Intuitively, it delays the execution of each job up to a point where any further delay would cause some job in the system to miss its deadline. We fix the delays of jobs in increasing order of (harmonic) period length, and then the delay for each job of the same task is identical. In a sense, the resulting schedule is a “reversed” RM-schedule (see Figure 2). Even though this strategy might seem counterintuitive, we show that this schedule is optimal, meaning that all jobs meet their deadlines if and only if the task system is feasible. Most importantly, the schedule can be described compactly and we can construct it (implicitly) in polynomial time.

We round up our results by giving a simpler EDF-schedulability test for the case that all period lengths *and* relative deadlines of the task system are jointly harmonic (Section IV-A).

B. Related work

1) Fixed priority: Many known results for fixed priority systems assume that priorities are either in rate-monotonic priority order or in deadline-monotonic order. If the deadline of each task equals its period, then the two orders coincide and the seminal results of Serlin [33] and Liu and Layland [28] showed that for synchronous tasks (i.e., with a common release date), the RM (and DM) ordering is optimal. Liu and Layland [28] also provided a simple, sufficient, utilization-based schedulability test for implicit-deadline tasks. If the task set has constrained deadlines, then it is known that DM is optimal among the family of fixed priority preemptive algorithms for scheduling on a uniprocessor [27].

Exact schedulability tests have been proposed by Joseph and Pandya [22] and Audsley et al. [2]. In [24] the authors propose an alternative method of determining exact schedulability conditions, which also allows an average case analysis.

A significant research effort has been devoted to improve the running time of exact schedulability tests or to find good approximations of the response time that allow to derive

[2]This result seems to be known as folklore, see, for example, the surveys [4], [25], but it may be attributed to [24].

sufficient schedulability tests or to extend known results to more general system models. We refer, for example, to [3], [7], [8], [10], [19], [36] and references therein. Additionally, polynomial-time approximation schemes (PTAS) for DM-schedulability have been proposed in [17], [31] that – in some cases – are based on approximating the response time to any degree of accuracy.

The power of harmonic task systems or those that come close to them has been supported in [9], [16], [23]. The authors of [23] propose the harmonic chain method, which extends the Liu and Layland bounds by determining and exploiting harmonic relationships among periods. Their experiments give evidence that (nearly) harmonic task sets can utilize the processor more efficiently than arbitrary task sets. In [19] the authors study schedulability methods that are based on transforming a given task set $\mathcal{T}$ into another task set $\mathcal{T}'$ with harmonic periods, whose schedulability implies the schedulability of $\mathcal{T}$. In [32] it is shown that the assumption of harmonic periods yields more accurate upper bounds on the response times.

One result that gives a formal proof of the power of harmonic periods is the aforementioned exact RM-schedulability test for implicit-deadline uniprocessor task systems, which requires only to verify that the total utilization of the task set is at most 1. This result is useful also in a multiprocessor environment. Consider partitioned RM-scheduling, where the task set is divided into subsets, one for each processor, and then RM-scheduling is applied on each individual processor. In case of implicit deadlines, partitioned scheduling can be reduced to a makespan minimization problem on multiple processors. Then the above utilization-based test, combined with a PTAS [20] or FPTAS [21] for the makespan minimization problem, gives nearly exact tests for partitioned multiprocessor RM-schedulability.

2) EDF: We already observed that existing results on exact schedulability analysis for EDF need to compute the demand bound function of the task set at several appropriate time intervals. Given the importance of the problem, there has been a significant effort in order to reduce the number of time intervals to be checked while guaranteeing a good approximation of the demand function. Many such sufficient schedulability tests have been proposed; we refer, for example, to [5], [6], [11], [26], [38] and references therein. Moreover, a fully polynomial-time approximation scheme has been proposed by Albers and Slomka [1]. However, the running time of known exact algorithms is not satisfactory; for example, in [38] it has been observed that *"the significant effort required to perform the exact schedulability test restricts the use of EDF in realistic systems; hence, the EDF algorithm has not been used as widely as the fixed priority algorithms in commercial realtime systems"*.

II. System model and notation

We consider a hard real-time system comprising a set $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of independent real-time sporadic tasks, each task consisting of a possibly infinite sequence of jobs which must be completed before their deadlines. Jobs of a sporadic task τ_j may arrive irregularly, but have a minimum inter-arrival time, also known as period. Each task τ_j is defined by a tuple (c_j, d_j, p_j), consisting of a *worst-case execution time* c_j, a *period* p_j, and a *relative deadline* d_j, with $c_j \le d_j \le p_j$ (*constrained deadlines*). We refer to [4] for the precise semantics of the model. We assume that all input parameters are integral and positive.

We restrict ourselves to task sets with *harmonic periods*, which means that for all $i, j = 1, \ldots, n$, either $p_i \mid p_j$ (p_i divides p_j) or $p_j \mid p_i$ (p_j divides p_i).

In our analysis, we will use two further assumptions that come without loss of generality:

1) the execution requirement of each job of task τ_j always equals its worst-case execution time;
2) the job sequence is the *synchronous arrival sequence* (*SAS*), that is, the i-th job of each task τ_j is released at time $(i-1)p_j$ and due at time $(i-1)p_j + d_j$.

We remark that both assumptions are without loss of generality: the first one, because we only consider *predictable* scheduling algorithms (in the sense of Ha and Liu [18]); the second one, because it is known (see for example [4, Lemma 28.9, 28.10]) that the SAS is the worst-case arrival sequence for determining response times and EDF-schedulability.

The *hyperperiod* P of task set $\mathcal{T}$ is defined as the least common multiple of the period lengths of the tasks in $\mathcal{T}$; since periods are harmonic, we have $P = \max_{i=1}^{n} p_i$.

The *utilization* of a task τ_i is the quantity c_i/p_i. The utilization $U(\mathcal{T})$ of task set $\mathcal{T}$ is $\sum_{i=1}^{n} c_i/p_i$. If $U(\mathcal{T}) > 1$, then there are job sequences of $\mathcal{T}$ that cannot be feasibly scheduled by any algorithm; hence, in the remainder of the paper we will use the assumption $U(\mathcal{T}) \le 1$.

III. Fixed priority scheduling

Given an arbitrary fixed priority order (a total ordering of all tasks), in this section we assume that the tasks are indexed according to this priority order, that is, a task τ_i has higher priority than task τ_j if and only if $i < j$. In a given schedule according to the given priority order, the *response time* of a job is the time that elapses between its release time and its completion time. The response time r_j of a task τ_j is the maximum response time that a job of this task may incur.

Consider a task system $\mathcal{T}$ with a given fixed priority order. Recall that without loss of generality we can consider the synchronous arrival sequence (SAS) of jobs. In the SAS, the response time of the first job of task τ_j is the largest among all the jobs of τ_j [4], [22], [37]. Hence, in order to check whether the given system is feasible, it is sufficient to compute for each task τ_j the response time, r_j, of the first job of τ_j in the SAS, and to determine whether $r_j \le d_j$ for each $j = 1, \ldots, n$.

It is known that r_j is the minimum $t > 0$ that satisfies the following equality:

$$c_j + \sum_{i<j} \left\lceil \frac{t}{p_i} \right\rceil \cdot c_i = t. \tag{1}$$

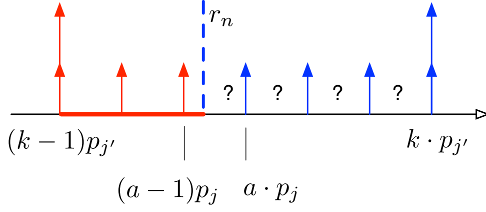

Fig. 1. An illustration of Lemma 2. Condition (2) is satisfied (respectively, not satisfied) at the blue (resp., red) time points. The condition may or may not be satisfied at the time points marked by question marks. (Red points are on the left of r_n; blue points are on the right of r_n and are marked by an upward arrow.)

In the sequel we also use the following equivalent definition: r_j is the minimum $t > 0$ that satisfies the following inequality:

$$c_j + \sum_{i<j} \left\lceil \frac{t}{p_i} \right\rceil \cdot c_i \leq t. \tag{2}$$

We now show how to compute the response time of a task in polynomial time if the periods are harmonic. Since the response time depends only on the schedule of higher-priority tasks (see Condition (2)), we can assume without loss of generality that we are interested in the response time of the task with lowest priority, task τ_n. Firstly, we observe that when periods are harmonic, the hyperperiod of the tasks in $\mathcal{T} \setminus \{\tau_n\}$ is certainly not larger than $P := \max_{i=1}^{n} p_i$, and thus, the schedule for these tasks is the same in each interval $[(q-1)P, qP)$ for any $q \in \mathbb{N}$. When $U(\mathcal{T}) \leq 1$, it is easy to show that the response time of τ_n is not larger than $P \cdot c_n$. We give a short proof for completeness.

Lemma 1. *If $U(\mathcal{T}) \leq 1$, then Condition (2) is satisfied with $j = n$ and $t = P \cdot c_n$ and, therefore, $r_n \leq P \cdot c_n$.*

Proof: The left hand side of (2) with $j = n$ and $t = P \cdot c_n$ can be bounded as follows:

$$c_n + \sum_{i<n} \left\lceil \frac{Pc_n}{p_i} \right\rceil \cdot c_i = c_n + \sum_{i<n} \frac{c_i}{p_i} \cdot Pc_n \leq \sum_{i \leq n} \frac{c_i}{p_i} \cdot Pc_n \leq Pc_n,$$

where the equality uses the fact that periods are harmonic, the first inequality uses the fact that $P/p_n \geq 1$ and $c_n \geq 1$, and the second inequality uses $U(\mathcal{T}) \leq 1$. ■

The following lemma is crucial for our approach to computing the response time. It allows to reduce the search to *release intervals*, that is, the intervals between two consecutive job arrivals of the same task. Informally speaking, the lemma says the following: Suppose the response time r_n lies in the release interval I of some task $\tau_{j'}$, and let p_j be the largest period that is less than $p_{j'}$. Then Condition (2) is also satisfied for every integer multiple of p_j equal or larger than r_n within I. Thus, r_n is at most the smallest such multiple of p_j. See Figure 1 for an illustration of the lemma.

Lemma 2. *Let $\mathcal{T}$ be a task system with total utilization at most 1. Assume $r_n \in ((q-1) \cdot p_{j'}, q \cdot p_{j'}]$ for some $q \in \mathbb{N}$, and let p_j be the largest period that is less than $p_{j'}$. If all periods are harmonic, then $t(a) := a \cdot p_j$ satisfies Condition (2) for any $a \in \mathbb{N}$ such that $t(a) \in [r_n, q \cdot p_{j'}]$.*

Proof: Consider some time point $t(a) = a \cdot p_j$, $a \in \mathbb{N}$, such that $r_n \leq a \cdot p_j \leq q \cdot p_{j'}$. We need to show that

$$c_n + \sum_{i<n} \left\lceil \frac{a \cdot p_j}{p_i} \right\rceil \cdot c_i \leq a \cdot p_j.$$

First, we observe that for all indices $i < n$ with $p_i > p_j$ we have that $\lceil a \cdot p_j / p_i \rceil = \lceil r_n / p_i \rceil$. The reason is that with harmonic periods and by definition, $a \cdot p_j$ and r_n are within the same release interval $((q-1) \cdot p_{j'}, q \cdot p_{j'}]$ of task $\tau_{j'}$, and thus within the same release interval $((q'-1) \cdot p_i, q' \cdot p_i]$ of any task τ_i with $i < n$ and $p_i > p_j$ (for a suitable value q'). Furthermore, for all $i < n$ with $p_i \leq p_j$ the ratio $a \cdot p_j / p_i$ is integral since $p_i \,|\, p_j$. Hence, the left-hand side of inequality (2) evaluated at $t(a)$ equals

$$\begin{aligned}
& c_n + \sum_{i<n} \left\lceil \frac{a \cdot p_j}{p_i} \right\rceil \cdot c_i \\
&= c_n + \sum_{\substack{i<n \\ p_i > p_j}} \left\lceil \frac{r_n}{p_i} \right\rceil \cdot c_i + \sum_{\substack{i<n \\ p_i \leq p_j}} \frac{a \cdot p_j}{p_i} \cdot c_i \\
&\leq c_n + \sum_{i<n} \left\lceil \frac{r_n}{p_i} \right\rceil \cdot c_i + \sum_{\substack{i<n \\ p_i \leq p_j}} \frac{a \cdot p_j - r_n}{p_i} \cdot c_i \\
&= \left(c_n + \sum_{i<n} \left\lceil \frac{r_n}{p_i} \right\rceil \cdot c_i \right) + (a \cdot p_j - r_n) \cdot \sum_{\substack{i<n \\ p_i \leq p_j}} \frac{c_i}{p_i} \\
&\leq r_n + (a \cdot p_j - r_n) \ = \ a \cdot p_j \, .
\end{aligned}$$

The last inequality holds since by definition inequality (2) is satisfied for $t = r_n$, and due to the utilization bound $\sum_{i=1}^{n} c_i / p_i \leq 1$. ■

The lemma suggests the following algorithm for computing the response time r_n. By Lemma 1, we know that r_n lies in the interval $(LB, UB]$ with $LB = 0$ and $UB = P \cdot c_n$. Using Lemma 2, we iteratively reduce the search interval $(LB, UB]$. Beginning with the largest period P, we find the least integer $a \in (LB/P, UB/P]$ such that Condition (2) is satisfied at $a \cdot P$. By Lemma 2 we can increase LB to $(a-1) \cdot P$ and reduce UB to $a \cdot P$. We continue with the longest period length that is shorter than P and so on, until the shortest period has been considered. At this point we have $UB - LB = \min_{i=1}^{n-1} p_i$ and we are able to determine r_n exactly. More formally, the description is in given in Algorithm 1.

Theorem 1. *Algorithm 1 correctly computes the response time r_n of task τ_n in time $\mathcal{O}(n \log n + n \log P)$.*

Proof: The correctness of the algorithm hinges on Lemmata 1 and 2. First of all, note that since $U(\mathcal{T}) \leq 1$ then Lemma 1 implies $r_n \leq P \cdot c_n$. Then the **for** loop of the algorithm maintains the invariant that the values of LB

Algorithm 1 Exact response time computation

Input: A constrained-deadline sporadic task system $\mathcal{T}$ with harmonic periods s.t. $U(\mathcal{T}) \leq 1$ and an FP ordering $(\tau_1, \tau_2, \ldots, \tau_n)$ of the tasks.

Output: Response time r_n.

1: Let $\pi : \{1, 2, \ldots, n-1\} \to \{1, 2, \ldots, n-1\}$ be a permutation that orders tasks by non-increasing period lengths, i.e., $i < j$ implies $p_{\pi(i)} \geq p_{\pi(j)}$.

2: $LB \leftarrow 0$

3: $UB \leftarrow c_n \cdot P$

4: **for** $i \leftarrow 1$ to $n-1$ **do**

5: Use binary search to find the least integer a such that a is in the interval $(LB/p_{\pi(i)}, UB/p_{\pi(i)}]$ and satisfies (2).

6: $LB \leftarrow (a-1) \cdot p_{\pi(i)}$

7: $UB \leftarrow a \cdot p_{\pi(i)}$

8: **end for**

9: $r_n \leftarrow c_n + \sum_{i<n} \left\lceil \frac{UB}{p_i} \right\rceil \cdot c_i$

10: **return** r_n

and UB are such that $LB < r_n \leq UB$; more precisely, Condition (2) is guaranteed to be violated for $t = LB$ and to be satisfied for $t = UB$. This is obviously true when LB is initialized to 0 and, by Lemma 1, when $UB = c_n \cdot P$. Then, at each iteration i, LB is updated to a value $(a-1) \cdot p_{\pi(i)}$ which does not satisfy (2), due to the minimality of a. Lemma 2, on the other hand, ensures that the updated UB does satisfy Condition (2). Notice also that, at the end of each iteration of the **for** loop, Condition (2) is satisfied by $t = UB$ and not satisfied by $t = LB$. Since the periods are harmonic and thus $p_{\pi(i)}$ always divides UB, the binary search procedure is successful at each iteration, because $UB/p_{\pi(i)}$ is always a possible target value of the binary search.

At the end of the **for** loop, the interval $(LB, UB]$ is such that $UB - LB = \min_{i=1}^{n-1} p_i$. Therefore, the terms $\lceil r_n/p_i \rceil$ and $\lceil UB/p_i \rceil$ are equal for each $i < n$. This implies that

$$r_n = c_n + \sum_{i<n} \left\lceil \frac{r_n}{p_i} \right\rceil \cdot c_i = c_n + \sum_{i<n} \left\lceil \frac{UB}{p_i} \right\rceil \cdot c_i,$$

which justifies the last step of the algorithm.

Concerning the running time, observe that apart from the initial ordering step (Step 1, which requires time $\mathcal{O}(n \log n)$), it is dominated by the overall cost of checking Condition (2) in the main loop. Since the cost of checking Condition (2) is $\mathcal{O}(n)$, it follows that the overall cost of Algorithm 1 amounts to a number of arithmetic operations that is bounded by $\mathcal{O}(n \log n + n \cdot n_T)$, where n_T denotes the total number of times that Condition (2) is tested. We now show that $n_T = \log(c_n \cdot P)$.

First observe that when $i = 1$ the algorithm performs a binary search for a multiple of P between $LB = 0$ and $UB = c_n P$. Hence, in the worst case, there are at most $\log(c_n P/P) = \log c_n$ points to be checked. When $i > 1$, in the i-th iteration of the main loop the algorithm runs a binary search procedure which tests integer multiples of the current period $p_{\pi(i)}$ on an interval of length at most $UB - LB = p_{\pi(i-1)}$. This implies that condition (2) is tested at most $\log(p_{\pi(i-1)}/p_{\pi(i)})$ times. Summing across i, we obtain that

$$\begin{aligned} n_T &= \log c_n + \sum_{i=2}^{n} \log(p_{\pi(i-1)}/p_{\pi(i)}) \\ &\leq \log c_n + \log P = \log(c_n \cdot P). \end{aligned}$$

Therefore, the running time of Algorithm 1 is bounded by $\mathcal{O}(n \log n + n \cdot \log(c_n \cdot P)) = \mathcal{O}(n \log n + n \log P)$ (since $c_n \leq p_n \leq P$). ■

To test the schedulability of a given fixed priority order, it suffices to verify whether the response time of every task satisfies $r_i \leq d_i$. Thus, by Theorem 1 we can simply execute Algorithm 1 for every task.

Corollary 1. *There is a polynomial time algorithm that, on input a constrained-deadline harmonic task set $\mathcal{T}$ and a fixed priority order, decides whether $\mathcal{T}$ is* FP*-schedulable under the given order on one processor.*

IV. Dynamic Scheduling – EDF-Schedulability

First, we present a simple and fast schedulability test for the special case when deadlines and periods are jointly harmonic. Then we present our main result: a polynomial-time algorithm for testing EDF-schedulability of tasks with harmonic periods.

We will need some more job-specific notation: Let $i(j)$ denote the i-th job of task τ_j. We denote its arrival date as $a_{i(j)}$ and its absolute deadline as $\bar{d}_{i(j)} := a_{i(j)} + d_j$. If the task to which a job belongs is clear from the context or irrelevant, then we omit the reference to the task index. Vice versa, we let $\tau(i)$ denote the task that released job i; we also say that $\tau(i)$ *owns* job i.

A. A test for periods and deadlines that are jointly harmonic

In this subsection we consider task systems in which the deadlines and periods of the tasks are jointly harmonic, that is, for all $x_i, x_j \in \{d_1, \ldots, d_n, p_1, \ldots, p_n\}$, either $x_i | x_j$ or $x_j | x_i$. We say such a task system is *fully harmonic*.

Definition 1. Two jobs i and k are a *strictly crossing pair* if $a_i < a_k < \bar{d}_i < \bar{d}_k$.

Lemma 3. *There is no strictly crossing pair of jobs in the synchronous arrival sequence of a fully harmonic constrained-deadline task system.*

Proof: We give a proof by contradiction. Suppose there exist two jobs i and k with $a_i < a_k < \bar{d}_i < \bar{d}_k$. First observe that the periods of the tasks owning jobs i and k satisfy $p_{\tau(k)} < p_{\tau(i)}$, because otherwise $p_{\tau(i)} \,|\, p_{\tau(k)}$ and thus the arrival of i would coincide with the arrival of k, a contradiction to $a_i < a_k$. By the same argument, we see that $p_{\tau(k)} \,|\, a_i$, and at a_i there is released another job $k' \neq k$ of task $\tau(k)$ which must have its absolute deadline at or before a_k (recall that deadlines are constrained) and thus before $\bar{d}_i$. Hence, $p_{\tau(k)} \leq \bar{d}_i - a_i = d_{\tau(i)}$, which implies $p_{\tau(k)} \,|\, \bar{d}_i$, because $p_{\tau(k)} \,|\, d_{\tau(i)}$ (periods and deadlines are jointly harmonic) and $p_{\tau(k)} \,|\, a_i$ (see above).

In a synchronous arrival sequence, jobs are released as early as possible and thus jobs of task $\tau(k)$ are released at every integer multiple of $p_{\tau(k)}$; in particular, there arrives a job k'' of task $\tau(k)$ at time $\bar{d}_i$, and thus, k itself must have finished by that time, i.e., $\bar{d}_k \leq \bar{d}_i$, which gives a contradiction and concludes the proof. ■

Lemma 4. *Consider the synchronous arrival sequence of a fully harmonic task system that is not EDF-schedulable, and let τ_j be the task with the earliest deadline miss. Then EDF fails at time d_j, the absolute deadline of τ_j's first job.*

Proof: Let $i(j)$ be the first job of task τ_j that misses its deadline $\bar{d}_i$ in an EDF schedule. Recall that a_i is the arrival time of job i.

We determine the workload that EDF assigns to the interval $[a_i, \bar{d}_i]$ and which delays the processing of i. We first observe that only jobs with an arrival time equal or larger than a_i (but before $\bar{d}_i$) can contribute to this workload. The reason is that by Lemma 3, jobs arriving before a_i have their absolute deadline before a_i or after $\bar{d}_i$, and in both cases they do not delay the execution of i in the EDF schedule. Furthermore, the jobs released after a_i with a deadline before $\bar{d}_i$ have, by the same reasoning as in the proof of Lemma 3, a period strictly less than p_j: indeed, if a job k had $a_k > a_i$ and $\bar{d}_k \leq \bar{d}_i$, then $p_{\tau(k)} < p_j$, otherwise $p_j \,|\, p_{\tau(k)}$ and the arrival of i would coincide with a_k. We conclude that the workload that delays the processing of job i in its time window $[a_i, \bar{d}_i]$ is only due to

- jobs with arrival time a_i, and
- jobs with arrival time in $(a_i, \bar{d}_i)$ that belong to tasks with period less than p_j.

In a synchronous arrival sequence, these are exactly the same jobs that delay the first job released by task τ_j. Thus, if $i(j)$ fails to complete, then the first job released by τ_j also fails to complete within its time interval $[0, d_j]$, which concludes the proof. ■

Algorithm 2 EDF-schedulability test for fully harmonic tasks

Input: A task system $\mathcal{T}$ with harmonic periods and deadlines.
Output: YES/NO-decision about $\mathcal{T}$ being EDF-schedulable.

1: **for** $\tau_i \in \mathcal{T}$ **do**
2: **if** $\sum_{\tau_k \in \mathcal{T}} \lfloor (d_i + p_k - d_k)/p_k \rfloor \cdot c_k > d_i$ **then**
3: **return** NO
4: **end if**
5: **end for**
6: **return** YES

Theorem 2. *There is a polynomial-time algorithm that decides whether a fully harmonic constrained-deadline task set is EDF-schedulable on one processor.*

Proof: A constrained-deadline task system is EDF-schedulable if and only if its synchronous arrival sequence is EDF-schedulable [4, Lemma 28.9]. For such an arrival sequence, Lemma 4 states that it is sufficient to test for each task τ_i the deadline of its first occurrence, that is, the time d_i. Thus, our algorithm (Algorithm 2) does the following: for $i = 1, \ldots, n$, test if EDF fails at time d_i. By [5, Lemma 4.2] this can be done by evaluating the demand bound function at time $t = d_i$, that is, we check if

$$\sum_{\tau_k \in \mathcal{T}} \left\lfloor \frac{d_i + p_k - d_k}{p_k} \right\rfloor \cdot c_k \leq d_i \,. \tag{3}$$

If all such tests are satisfied, then $\mathcal{T}$ is EDF-schedulable; otherwise, we have found a time t by which EDF fails.

Regarding the running time, observe that for each $i = 1, 2, \ldots, n$, Condition (3) can be tested in time linear in the input size. Therefore, the overall complexity of the test is quadratic. ■

B. A polynomial-time test for harmonic periods

Consider a task system $\mathcal{T}$ with harmonic periods and arbitrary (but constrained) deadlines. We present a polynomial time algorithm which tests whether for every job sequence of $\mathcal{T}$ a feasible uniprocessor schedule exists (and thus whether the task set is EDF-schedulable [11], [28]).

Recall from Section II that it is sufficient to verify the schedulability of a synchronous arrival sequence in which all jobs attain their WCETs. Without loss of generality, assume that the tasks in $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$ are sorted non-decreasingly by period length, that is, $p_j \leq p_{j'}$ for any j, j' with $j < j'$. To prove the result we will proceed in three steps. Namely,

1) We define a (non-EDF) *procrast schedule* S for the synchronous arrival sequence of $\mathcal{T}$; S is implicitly defined by a sequence of values $b_1, b_2, \ldots, b_n$ (one per task).
2) We then present Algorithm 3, which either constructs S or asserts that $\mathcal{T}$ is infeasible; we show that the running time of Algorithm 3 is polynomial in the input size.
3) Finally, we show that if Algorithm 3 is successful then S is a feasible schedule for the SAS, while if Algorithm 3 fails then $\mathcal{T}$ is infeasible (even for EDF).

1) Definition of procrast schedule S: We define a schedule S for the time interval $[0, p_n)$. This is sufficient for describing the full schedule, since we assume the period lengths to be harmonic, and thus, p_n equals the hyperperiod of all period lengths in the instance.

Schedule S is defined by considering tasks in order; we denote by S_j, $j = 1, 2, \ldots, n$, the partial schedule of jobs released by tasks $\tau_1, \tau_2, \ldots, \tau_j$. Algorithm 3 starts from the empty schedule S_0 and, at each iteration $j = 1, 2, \ldots, n$, defines S_j based on S_{j-1} by specifying how τ_j's jobs are scheduled. Namely, at iteration j, the algorithm delays the processing of each job of τ_j as much as possible given the execution requirements of all jobs released by shorter-period tasks specified by S_{j-1}. We refer to the schedule also as a *procrast schedule*. In particular, in a procrast schedule each job of task τ_1 (the shortest-period task) is scheduled for c_1 time units just before its deadline. Jobs of τ_j, $j = 2, \ldots, n$, are scheduled similarly; however, for these jobs we have to

take into account S_{j-1} and, in particular, the busy times in which the processor is executing jobs of $\tau_1, \ldots, \tau_{j-1}$.

To define S formally, it is sufficient to specify the time instant at which each job starts execution. In fact, at any point in time t there can be several pending jobs (i.e., jobs that have started execution but that have not been completed), each one released by a different task; among this set of jobs, at time t schedule S executes the one released by the task τ_i having the least index i.

The critical observation is that the time instants in which each job starts execution follow a regular pattern. In the following we define a value b_j for each task τ_j and use it for all jobs released by τ_j: job $i(j)$ of τ_j, which is released at time $a_{i(j)}$, starts execution at time $a_{i(j)} + b_j$. At time t, S executes job $i(j)$ if $t \geq a_{i(j)} + b_j$, $i(j)$ is not completed and if no job released by a higher priority (shorter period) task is pending. In fact, the resulting schedule can be seen as a fixed priority schedule, where the priorities are assigned to the tasks according to their period lengths and the release of each job $i(j)$ is moved to $a_{i(j)} + b_j$. Intuitively, $a_{i(j)} + b_j$ is the maximum amount of time by which we may delay the execution of job $i(j)$ without missing its deadline, given the schedule of tasks $\tau_1, ..., \tau_{j-1}$. For this reason, we call b_j the *panic offset* of task τ_j.

We give details on how to compute panic offsets later. For illustrative purposes, consider task τ_1 (the shortest period task). We have $b_1 = d_1 - c_1$. Clearly, for each job $i(1)$ released by τ_1 we have that $a_{i(1)} + b_1 = a_{i(1)} + d_1 - c_1 = \bar{d}_{i(1)} - c_1$ is the latest point in time to start $i(1)$, since otherwise the deadline of $i(1)$ would be missed. We define a schedule S_1 for the jobs of τ_1 by executing each job $i(1)$ of τ_1 during the interval $[a_{i(1)} + b_1, a_{i(1)} + d_1)$.

Definition 2 (Procrast schedule S_j)**.** A schedule is called *procrast schedule* S_j *with panic offsets* $b_1, ..., b_j$ (or only *procrast schedule* S_j for short) if it satisfies the following properties:

1) each job $i(1)$ of τ_1 is executed for c_1 time units during the interval $[a_{i(1)} + b_1, a_{i(1)} + d_1)$;
2) each job $i'(j')$ of task $\tau_{j'}$, $j' = 2, ..., j$, is executed for $c_{j'}$ time units during the interval $[a_{i'(j')} + b_{j'}, a_{i'(j')} + d_{j'})$ when no job created by some task $\tau_{j''}$, $j'' < j'$, is running;
3) for each job $i'(j')$ of $\tau_{j'}$, at each time $t \in [a_{i'(j')} + b_{j'}, a_{i'(j')} + d_{j'})$ the processor is never idle and processes a job of a task in $\{\tau_1, \ldots, \tau_{j'}\}$.

Observe that the panic offsets $b_1, \ldots, b_j$ are sufficient to define the procrast schedule S_j for the task set $\{\tau_1, \ldots, \tau_j\}$. We let $\mathsf{S} := \mathsf{S}_n$ and we observe that, by 1) and 2) in the above definition, a procrast schedule is a feasible schedule. Also notice that each job $i'(j')$ of task $\tau_{j'}$ runs in particular during the interval $[a_{i'(j')} + b_{j'}, a_{i'(j')} + b_{j'} + 1)$; this fact will be used in the sequel.

Example 1. Consider $\mathcal{T} = \{\tau_1, \tau_2, \tau_3\}$ with $\tau_1 = (1, 3, 4)$, $\tau_2 = (3, 5, 8)$, $\tau_3 = (3, 10, 16)$. Figure 2 shows the procrast schedule S_3; the panic offsets are $b_1 = 2, b_2 = 1, b_3 = 5$.

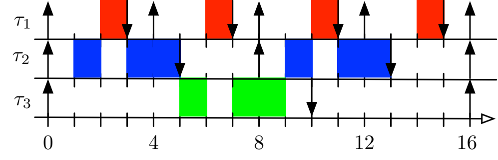

Fig. 2. The procrast schedule for the three tasks of Example 1.

2) Computation of panic offsets: We now present Algorithm 3, which constructs the procrast schedule S for task set $\mathcal{T}$ by computing panic offsets $b_1, \ldots, b_n$; if the algorithm does not succeed, then it asserts that $\mathcal{T}$ is infeasible. We will also prove that the running time of the algorithm is polynomial.

The algorithm sorts tasks by non-decreasing period lengths and then computes S inductively. Namely, it initializes S_0 to be the empty schedule; at each iteration, given a feasible procrast schedule S_{j-1} with $b_1, ..., b_{j-1}$, it computes the panic offset b_j to obtain the procrast schedule S_j. We define schedule S_1 by the panic offset $b_1 := d_1 - c_1$. Observe that this is a feasible procrast schedule. Now consider some integer $j \geq 2$ and suppose there is a feasible procrast schedule S_{j-1} for all tasks $\tau_1, ..., \tau_{j-1}$ satisfying the properties of Definition 2. By this definition, the busy times of the processor are exactly $\cup_j \cup_i [a_{i(j)} + b_j, a_{i(j)} + d_j)$. For finding b_j we have to compute the largest value $x \geq 0$ such that during $[x, d_j)$ the idle time of the processor equals exactly c_j. Then we set $b_j := x$. If there is no such non-negative value x, then we output that the task system is infeasible.

Algorithm 3 Computation of Procrast Schedule S

Input: A task system $\mathcal{T}$ with harmonic periods
Output: Procrast Schedule S or assertion that $\mathcal{T}$ is infeasible

```
1:  sort tasks T such that p_1 ≤ p_2 ≤ ... ≤ p_n
2:  set S_0 to be empty schedule
3:  for j ← 1 to n do
4:     if idle time during [0, d_j) is strictly less than c_j then
5:        return  task set T infeasible
6:     else
7:        compute largest x ≥ 0 s.t. S_{j-1} has exactly c_j units
          of idle time in [x, d_j)
8:        b_j ← x
9:        Let S_j be the procrast schedule defined by panic
          offsets b_1, ..., b_j
10:    end if
11: end for
12: S ← S_n
13: return  S
```

In reference to Algorithm 3, we now show how to perform the test of Line 4 and how to compute the value x (Line 7) through a polynomial time subroutine that computes the idle time in S_{j-1} during the interval $[x, d_j)$ for a given value x.

Lemma 5. *Consider a feasible procrast schedule S_{j-1} for tasks $\{\tau_1, ..., \tau_{j-1}\}$, specified by panic offsets $b_1, ..., b_{j-1}$, and a value $x \geq 0$. The amount of idle time in S_{j-1} during $[x, d_j)$ can be computed in polynomial time.*

Before proving the lemma we show how it allows to complete the description of Algorithm 3. First of all observe that Lemma 5 allows to check whether during $[0, d_j)$ the amount of idle time is at least c_j (Line 4 of Algorithm 3). If this is not the case then Algorithm 3 outputs that the system is infeasible and we stop. Otherwise, we use Lemma 5 several times to compute the largest value $x \geq 0$ such that during $[x, d_j)$ the idle time of the processor equals exactly c_j (see Line 7 in the description of Algorithm 3). We do this with a binary search procedure. First, we define a lower bound $LB := 0$ and an upper bound $UB := d_j$. We keep as an invariant that the idle time within $[LB, d_j)$ is at least c_j and the idle time within $[UB, d_j)$ is at most c_j. In each iteration, we check the idle time within $[(UB-LB)/2, d_j)$ using Lemma 5 and update LB or UB accordingly. Hence, after $O(\log(d_j))$ iterations we find the largest value x with the desired property. We define the panic offset of task τ_j to be $b_j := x$ and continue with task τ_{j+1}.

The proof of Lemma 5 hinges on the following lemma that computes in S_{j-1} the latest point in time $t' \in \mathbb{N}$ before a given time $t \in \mathbb{N}$ such that no jobs are started that will not finish by time t'. Note that this is almost but not exactly the same as saying that the processor is idle at time t'. In particular, it might be the case that $t' \in [a_{i(j)} + b_j, a_{i(j)} + d_j)$ for some job $i(j)$ and no other job is pending. Formally, we have the following lemma.

Lemma 6. *Consider a feasible procrast schedule S_{j-1} for tasks $\{\tau_1, ..., \tau_{j-1}\}$, specified by panic offsets $b_1, ..., b_{j-1}$, and a value $t \in \mathbb{N}$. There is a polynomial time algorithm which computes the largest value $t' \leq t$ such that $t' \notin (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ for each task $\tau_{j'}$ with $j' \in \{1, ..., j-1\}$ and each job $i(j')$.*

Proof: First, we consider the given value t and verify whether $t \in (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ for some job $i(j')$ of some task $\tau_{j'}$, $j' \in \{1, ..., j-1\}$. This can be done in polynomial time because if there is such a job then it has to be the job $i(j')$ with $t \in [a_{i(j')}, a_{i(j')+1})$ and its index $i(j')$ equals $\lfloor t/p_{j'} \rfloor$. Thus, for each task $j' \in \{1, ..., j-1\}$ we consider the job with index $\lfloor t/p_{j'} \rfloor$ and verify if $t \in (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$. If this is not the case for any $j' \in \{1, ..., j-1\}$, then we output $t' := t$ and we are done.

Consider now the case that $t \in (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ for some job $i(j')$. We define $t_0 := t$ and let $j^{(0)}$ be the index of a task such that $t_0 \in (a_{i(j^{(0)})} + b_{j^{(0)}}, a_{i(j^{(0)})} + d_{j^{(0)}})$ for some job $i(j^{(0)})$. We define $t_1 := a_{i(j^{(0)})} + b_{j^{(0)}} < t_0$. We iterate: If t_1 has the property stated in the lemma, that is, $t_1 \notin (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ for any job of any task $\tau_{j'}$ with $j' \in \{1, ..., j-1\}$, then we are done and define $t' := t_1$. (We verify this as described above for value t.) Otherwise there must be a task $j^{(1)}$ such that $t_1 \in (a_{i(j^{(1)})} + b_{j^{(1)}}, a_{i(j^{(1)})} + d_{j^{(1)}})$ for some job $i(j^{(1)})$ of $j^{(1)}$. In that case $j^{(0)} < j^{(1)}$, since by definition of procrast schedules at time t_1 a job of task $j^{(0)}$ is executed and during the whole interval $[a_{i(j^{(1)})} + b_{j^{(1)}}, a_{i(j^{(1)})} + d_{j^{(1)}})$ jobs of tasks having index in $\{1, ..., j^{(1)}\}$ are executed (note that $j^{(0)} \neq j^{(1)}$ since $d_j \leq p_j$ for all tasks τ_j). We iterate this procedure until we find a value t_k with the properties that we require for t'. Similarly as before, we can argue that $j^{(k'-1)} < j^{(k')}$ for all $k' \leq k$ and thus at most n iterations are needed. In every iteration we check for each task $\tau_{j'} \in \{\tau_1, ..., \tau_{j-1}\}$ whether it has a job $i(j')$ with $t_i \in (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$. By the above reasoning, it is sufficient to check this for the job with index $\lfloor t/p_{j'} \rfloor$. Thus, the overall running time of this subprocedure is bounded by $\mathcal{O}(n^2)$. The procedure is summarized in Algorithm 4. ■

We are now ready to prove Lemma 5.

Proof (of Lemma 5): The amount of idle time in $[x, d_j]$ equals the amount of idle time in $[0, d_j)$ minus the amount of idle time in $[0, x)$. Thus, it is sufficient to provide a polynomial-time routine for computing the amount of idle time for the interval $[0, x)$ for any value $x \geq 0$. We do this as follows.

Using the procedure from Lemma 6 we compute the last point in time $x' \leq x$ such that $x' \notin (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ for each task $j' \in \{1, ..., j-1\}$ and each job $i(j')$. By definition of procrast schedules, the processor is busy during the interval $[x', x)$. On the other hand, during the interval $[0, x')$ procrast executes only those jobs whose deadline is in $[0, x')$. Given x', we can easily compute for each task j' how many jobs of j' have their deadline in $[0, x')$. Hence, we can calculate the total execution time C of jobs i with $\bar{d}_i \in [0, x')$. Then, the amount of idle time within $[0, x)$ equals $x' - C$.

The overall procedure requires two calls to the procedure from Lemma 6 whose running time is bounded by $\mathcal{O}(n^2)$. Thus, the total running time of this procedure is also bounded by $\mathcal{O}(n^2)$. ■

3) Feasibility of the procrast schedule S (the feasibility test): The feasibility test for a given task system $\mathcal{T}$ now is as follows. We consider all tasks $\tau_j \in \mathcal{T}$ one after the other in order of their indices and run the algorithm for computing the panic offset b_j. If for some j there is no non-negative value b_j then we output that the system $\mathcal{T}$ is infeasible. Otherwise, we output that $\mathcal{T}$ is EDF-schedulable.

In the following two lemmas we show that both actions are justified. First, we argue that if $b_j \geq 0$ for a given feasible procrast schedule S_{j-1} for tasks $\{\tau_1, \dots, \tau_{j-1}\}$, then S_j is feasible. Secondly, we need to show that if $b_j < 0$ then the worst-case instance is infeasible.

Lemma 7. *If S_{j-1}, $j = 2, 3, \dots, n$, is a feasible procrast schedule and during $[0, d_j)$ the amount of idle time is at least c_j then also S_j is a feasible procrast schedule.*

Proof: Since the period lengths are harmonic, the hyperperiod of $\{p_1, ..., p_j\}$ equals p_j. Since we consider only the synchronous arrival sequence, if the first job of task τ_j meets its deadline then all jobs of τ_j meet their deadline. The former property is ensured due to the definition of b_j. In fact, $b_j \leftarrow x$

Algorithm 4 Computation of last time point t' before t without started job in procrast schedule S (Lemma 6)

Input: A procrast schedule S, specified by panic offsets $b_1, ..., b_{j-1}$ and a value t

Output: largest value $t' \leq t$ such that $t' \notin (a_{i(j')}+b_{j'}, a_{i(j')}+d_{j'})$ for each task $j' \in \{1, ..., j-1\}$ and each job $i(j')$

1: **if** $t' \notin (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ for each task $\tau_{j'}$ with $j' \in \{1, ..., j-1\}$ and each job $i(j')$ **then**
2: $\quad t' \leftarrow t$
3: $\quad$ **return** t'
4: **else**
5: $\quad t_0 \leftarrow t'$
6: $\quad k \leftarrow 0$
7: $\quad$ **while** there is a task $\tau_{j'}$ with $j' \in \{1, ..., j-1\}$ and a job $i(j')$ with $t_k \in (a_{i(j')} + b_{j'}, a_{i(j')} + d_{j'})$ **do**
8: $\quad\quad t_{k+1} \leftarrow a_{i(j')} + b_{j'}$
9: $\quad\quad k \leftarrow k + 1$
10: $\quad$ **end while**
11: **end if**
12: $t' \leftarrow t_k$
13: **return** t'

is executed in the j-th iteration of the main loop of Algorithm 3 only if there are at least c_j units of idle time in S_{j-1}. Hence, the schedule S_j is feasible. ■

Now we show that if our procedure does not give a feasible schedule, then no feasible schedule exists.

Lemma 8. *Given the procrast schedule S_{j-1} for the tasks $\{\tau_1, ..., \tau_{j-1}\}$ computed by the above procedure, if the amount of idle time during the interval $[0, d_j)$ is strictly smaller than c_j then task set $\{\tau_1, ..., \tau_j\}$ is infeasible.*

Proof: We show by induction the following slightly stronger claim. For each $j' \leq j-1$ the computed schedule $\mathsf{S}_{j'}$ maximizes the idle time during the interval $[0, x)$ for each $x \geq 0$ simultaneously. Formally, for each $j' \leq j-1$ and each $x \geq 0$ there is no feasible schedule for the tasks $\{\tau_1, ..., \tau_{j'}\}$ with strictly more idle time during $[0, x)$ than $\mathsf{S}_{j'}$.

We show the claim by induction over j'. For $j' = 1$ the claim is immediate since we start each job of τ_1 as late as possible. Now suppose that the claim is true for some value $j'-1$ and consider the schedule $\mathsf{S}_{j'}$ and a value $x \geq 0$. First assume that there is no job i of $\tau_{j'}$ such that $x \in [a_i, \bar{d}_i)$ and assume that there are exactly k jobs i' of $\tau_{j'}$ such that $\bar{d}_{i'} \leq x$. Then any feasible schedule has to work for $k \cdot c_{j'}$ units of time on jobs of $\tau_{j'}$ during $[0, x)$ and thus the claim follows from the induction hypothesis.

Now suppose that $x \in [a_{i'}, \bar{d}_{i'})$ for some job i' created by task $\tau_{j'}$. Since $p_{j''} | p_{j'}$ and $d_{j''} \leq p_{j''}$ for each $j'' \leq j'$ we can assume w.l.o.g. that $x \in [0, d_{j'})$, i.e., i' is the first job created by task $\tau_{j'}$. Suppose by contradiction that there is some feasible schedule $\bar{\mathsf{S}}$ for tasks $\{\tau_1, \ldots, \tau_{j'}\}$ with strictly more idle time during $[0, x)$ than $\mathsf{S}_{j'}$. W.l.o.g. we can assume that in $\bar{\mathsf{S}}$ between the time when i' is started and $d_{j'}$ the processor has no idle time, i.e., intuitively, job i' is started as late as possible in $\bar{\mathsf{S}}$. Let s denote the time when i' is started in $\bar{\mathsf{S}}$, i.e., executed for the first time, and recall that b_j is the respective time in $\mathsf{S}_{j'}$ (since i' is the first job of task $\tau_{j'}$). Observe that $\bar{\mathsf{S}}$ does not have any idle time during $[s, d_{j'})$. If $x \leq \min\{s, b_{j'}\}$ then we have a contradiction to the induction hypothesis as then during $[0, x)$ the schedules $\mathsf{S}_{j'}$ and $\mathsf{S}_{j'-1}$ are identical, in particular they do not execute a job from task τ_j, and they maximize the amount of idle time during $[0, x)$ among all feasible schedules for the tasks $\{\tau_1, ..., \tau_{j'}\}$.

Now suppose that $x > \min\{s, b_{j'}\}$. If $s \leq b_{j'}$ then in particular $x > s$ and also $\bar{\mathsf{S}}$ must have more idle time during $[0, s)$ than $\mathsf{S}_{j'}$ (using that $\bar{\mathsf{S}}$ does not have any idle time during $[s, d_j) \supseteq [s, x)$). However, during $[0, s)$ the schedules $\mathsf{S}_{j'}$ and $\mathsf{S}_{j'-1}$ are identical and do not execute a job from task τ_j (since $s \leq b_{j'}$) which contradicts the induction hypothesis using the same arguments as for the case that $x \leq \min\{s, b_{j'}\}$.

The last case is that $s > b_{j'}$, i.e., in $\bar{\mathsf{S}}$ the job i' starts later than in $\mathsf{S}_{j'}$. Denote by y the amount of idle time during $[0, d_{j'})$ in $\bar{\mathsf{S}}$ and by y' the amount of idle time during $[0, d_{j'})$ in $\mathsf{S}_{j'}$. As during $[b_{j'}, d_{j'})$ there is no idle time in $\mathsf{S}_{j'}$ we have that $y > y'$. However, if we take the schedule $\bar{\mathsf{S}}$ and remove all jobs from task τ_j' then we get a schedule $\bar{\mathsf{S}}'$ for the tasks $\{\tau_1, ..., \tau_{j'-1}\}$ whose idle time during $[0, d_{j'})$ equals $y + c_{j'}$. Likewise, the schedule $\mathsf{S}_{j'-1}$ has $y' + c_{j'}$ units of idle time during $[0, d_{j'})$. The induction hypothesis implies that $y + c_{j'} \leq y' + c_{j'}$ which implies that $y \leq y'$, a contradiction.

Finally, we show how the above claim implies the lemma. If in S_{j-1} the amount of idle time during the interval $[0, d_j)$ is strictly smaller than c_j then the latter has to hold for *any* feasible schedule for the tasks $\{\tau_1, ..., \tau_{j-1}\}$. Thus, there can be no feasible schedule for the tasks $\{\tau_1, ..., \tau_j\}$. ■

If, after completing all n iterations, the schedule S $(= \mathsf{S}_n)$ is feasible, then the task set $\mathcal{T}$ is feasible and therefore EDF-schedulable. This yields the following theorem.

Theorem 3. *There is a polynomial time algorithm that, on input a constrained-deadline harmonic task set $\mathcal{T}$, decides whether $\mathcal{T}$ is EDF-schedulable on one processor.*

Proof: We use Algorithm 3 to compute the procrast schedule S. If the routine fails then, according to Lemma 8, task set $\mathcal{T}$ is infeasible. On the other hand, if the routine is successful, then by Lemma 7 the resulting schedule is feasible and thus $\mathcal{T}$ is feasible (and therefore EDF-schedulable).

The main loop of Algorithm 3 runs in n iterations and in the j-th iteration the running time is dominated by the $\mathcal{O}(\log d_j)$ calls to the subroutine of Lemma 5, each having a running time of $\mathcal{O}(n^2)$. Thus, the overall running time is bounded by $\mathcal{O}(n^3 \cdot \log(\max_{j=1}^{n} d_j)) = \mathcal{O}(n^3 \cdot \log P)$, which is polynomially bounded in the input length. ■

V. Conclusion

Two of the most basic problems in real-time scheduling are computing the response time of a given task system and checking EDF-schedulability on a uniprocessor. It is known that both problems are computationally hard [13], [15] and

therefore, it is expected that no efficient algorithm exists. On the other hand, instances that arise in practice are often very special, in the sense that their period lengths are harmonic. In fact, harmonic period lengths allow high utilizations of the processors, which make them a natural choice for a system designer. For this important special class of instances, in this paper we presented the first efficient, exact polynomial time algorithms. This provides evidence that the computational complexity of the non-harmonic instances mostly stems from the possibly complicated algebraic structures in the period lengths.

Our results hold only for constrained-deadline task systems. Extending them to arbitrary (unconstrained) task systems is an interesting open question. Also, it would be interesting to extend our results to the study of exact schedulability tests for other algorithms, such as FPZL (Fixed Priority until Zero Laxity). Another important research topic is to extend our results to more general system models, including other parameters such as blocking times caused by shared resources. Finally, we observe that while our schedulability tests for fixed priority and fully harmonic EDF-schedulability are very efficient and applicable to large task sets, it would be interesting to improve the running time of the EDF-schedulability test for harmonic periods (Algorithm 3).

References

[1] K. Albers and F. Slomka. An event stream driven approximation for the analysis of real-time systems. In *Proc. 16th Euromicro Conf. on Real-Time Systems*, pages 187–195. IEEE, 2004.

[2] N. Audsley, A. Burns, M. Richardson, and A. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[3] S. K. Baruah. Efficient computation of response time bounds for preemptive uniprocessor deadline monotonic scheduling. *Real-Time Systems*, 47(6):517–533, 2011.

[4] S. K. Baruah and J. Goossens. Scheduling real-time tasks: Algorithms and complexity. In J. Y.-T. Leung, editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 28. CRC Press, 2003.

[5] S. K. Baruah, R. R. Howell, and L. E. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Comput. Sci.*, 118(1):3–20, 1993.

[6] 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 Systems*, 2:301–324, 1990.

[7] E. Bini, G. C. Buttazzo, and G. Buttazzo. Rate monotonic analysis: The hyperbolic bound. *IEEE Trans. Comput.*, 52(7):933–942, 2003.

[8] E. Bini, T. H. C. Nguyen, P. Richard, and S. K. Baruah. A response-time bound in fixed-priority scheduling with arbitrary deadlines. *IEEE Trans. Comput.*, 58(2):279–286, 2009.

[9] D. Chen, A. K. Mok, and T.-W. Kuo. Utilization bound revisited. *IEEE Trans. Comput.*, 52(3):351–361, 2003.

[10] R. I. Davis, A. Zabos, and A. Burns. Efficient exact schedulability tests for fixed priority real-time systems. *IEEE Trans. Computers*, 57(9):1261–1276, 2008.

[11] M. L. Dertouzos. Control robotics: The procedural control of physical processes. In *Proc. IFIP Congress*, pages 807–813, 1974.

[12] F. Eisenbrand, K. Kesavan, R. S. Mattikalli, M. Niemeier, A. W. Nordsieck, M. Skutella, J. Verschae, and A. Wiese. Solving an avionics real-time scheduling problem by advanced IP-methods. In *Algorithms – ESA 2010*, pages 11–22. Springer, 2010.

[13] F. Eisenbrand and T. Rothvoss. Static-priority real-time scheduling: Response time computation is NP-hard. In *Proc. IEEE Real-Time Systems Symp.*, pages 397–406. IEEE, 2008.

[14] F. Eisenbrand and T. Rothvoss. New hardness results for diophantine approximation. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – APPROX-RANDOM 2009*, pages 98–110. Springer, 2009.

[15] F. Eisenbrand and T. Rothvoss. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proc. 21st ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2010.

[16] M. Fan and G. Quan. Harmonic semi-partitioned scheduling for fixed-priority real-time tasks on multi-core platform. In *Proc. of DATE*, pages 503–508. IEEE, 2012.

[17] N. Fisher and S. K. Baruah. A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines. In *Proc. 17th Euromicro Conf. on Real-Time Systems*, pages 117–126. IEEE, 2005.

[18] R. Ha and J. W.-S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. In *Proc. 14th Int. Conference on Distributed Computing Systems*, pages 162–171, 1994.

[19] C.-C. Han and H.-Y. Tyan. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithm. In *Proc. IEEE Real-Time Systems Symp.*, pages 36–45. IEEE, 1997.

[20] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987.

[21] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM*, 23(2):317–327, 1976.

[22] M. Joseph and P. K. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[23] T.-W. Kuo and A. K. Mok. Load adjustment in adaptive real-time systems. In *Proc. IEEE Real-Time Systems Symp.*, pages 160–170. IEEE, 1991.

[24] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. IEEE Real-Time Systems Symp.*, pages 166–171. IEEE, 1989.

[25] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda. Fixed priority scheduling theory for hard real-time systems. In A. van Tilborg and G. Koob, editors, *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 1–30. Kluwer Academic, New York, 1991.

[26] J. Y.-T. Leung and M. L. Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Inf. Process. Lett.*, 11(3):115–118, 1980.

[27] J. Y.-T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Perform. Evaluation*, 2(4):237–250, 1982.

[28] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[29] A. K. Mok. *Fundamental design problems of distributed systems for the hard real-time environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[30] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos. Mixed-criticality real-time scheduling for multicore systems. In *10th IEEE International Conference on Computer and Information Technology, CIT 2010*, pages 1864–1871, 2010.

[31] T. H. C. Nguyen, P. Richard, and E. Bini. Approximation techniques for response-time analysis of static-priority tasks. *Real-Time Systems*, 43(2):147–176, 2009.

[32] C. Okwudire, M. van den Heuvel, R. Bril, and J. Lukkien. Exploiting harmonic periods to improve linearly approximated response-time upper bounds. In *Proc. ETFA*, pages 1–4. IEEE, 2010.

[33] O. Serlin. Scheduling of time critical processes. In *AFIPS Spring Joint Computing Conference*, pages 925–932, 1972.

[34] L. Sha, T. F. Abdelzaher, K.-E. Årzén, A. Cervin, T. P. Baker, A. Burns, G. C. Buttazzo, M. Caccamo, J. P. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2-3):101–155, 2004.

[35] L. Sha and J. B. Goodenough. Real-time scheduling theory and Ada. *IEEE Computer*, 23(4):53–62, 1990.

[36] M. Sjödin and H. Hansson. Improved response-time analysis calculations. In *Proc. IEEE Real-Time Systems Symp.*, pages 399–408. IEEE, 1998.

[37] K. Tindell, A. Burns, and A. J. Wellings. An extendible approach for analyzing fixed priority hard real-time tasks. *Real-Time Systems*, 6(2):133–151, 1994.

[38] F. Zhang and A. Burns. Schedulability analysis for real-time systems with EDF scheduling. *IEEE Trans. Comput.*, 58(9):1250–1258, 2009.