Preemptively Scheduling Hard-Real-Time Sporadic Tasks on One Processor *

Sanjoy K. Baruah, Aloysius K. Mok, and Louis E. Rosier

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

Abstract

In this paper, we consider the preemptive scheduling of hard-real-time sporadic task systems on one processor. We first give necessary and sufficient conditions for a sporadic task system to be feasible (i.e., schedulable). The conditions cannot, in general, be tested efficiently (unless **P** = **NP**). They do, however, lead to a feasibility test that runs in efficient pseudo-polynomial time for a very large percentage of sporadic task systems.

1 Introduction

Scheduling theory as it applies to hard-real-time environments — environments where the missing of a single deadline may have disastrous consequences — seems to currently be enjoying a renaissance. Hard-real-time scheduling problems may concern either fixed-duration tasks or recurring tasks that must be completed within a certain time frame. The problems most studied within the recurring category involve periodically recurring tasks [LL73, LM80, LM81, LW82, Mok83, BHR90]. Aperiodically or sporadically recurring tasks have also been studied [Mok83, LSS87, HL88, SLS88, JAM90, SSL89], but currently seem less understood. The main focus of this paper concerns *feasibility* testing with respect to sporadic task systems (i.e., testing whether a sporadic task system is schedulable).

Sporadic tasks were introduced by Mok [Mok83] to model external interrupts; i.e., events external to the computer system, such as a button being pushed or a train crossing a sensor. Clearly one cannot guarantee schedulability for interrupts that occur arbitrarily frequently. However, when an interval between successive invocations is guaranteed by the environment, schedulability becomes possible. A **sporadic task** is characterized by three positive integers — an *execution time* e, a *deadline* d (relative to the release time), and a *minimum separation* p, with $e \leq d$ and $e \leq p$. Sporadic tasks may make a request at any time, but two successive requests must be separated in time by at least p "time units." A request by T at time t requires that the processor be allocated (or scheduled) to T for e time units in the interval $[t, t+d)$. (We adhere to the discrete model of time where a time unit cannot be subdivided. The interval $[t, t+d)$ then is composed of d time units corresponding to times t, $t+1$, ..., $t+d-1$.) Note that d is allowed to exceed p. This is significant. For some applications, e.g., signal processing, it is possible to model the processing elements (nodes in a signal graph) as processes in our model. In this case, the deadline of a process may be larger than its period since there may be buffers which allow processing to be postponed past the end of the sampling period. A sporadic task system τ is a collection of sporadic tasks $\{T_1, \ldots, T_n\}$, with $T_i = (e_i, d_i, p_i)$, $1 \leq i \leq n$. τ is feasible (schedulable) iff it is possible for the processor to meet the requirements for all requests in each set of task requests that satisfies the separation constraints.

Liu & Layland [LL73] and Labetoulle [Lab74] first introduced an on-line deadline driven algorithm for scheduling periodic task systems. The algorithm is *optimal* in the sense that it will successfully generate a processor schedule for any periodic task system that can be scheduled. (If the system cannot be scheduled the algorithm will eventually miss a deadline.) Dertouzos [Der74] subsequently extended this to obtain a *deadline algorithm* that is optimal for sporadic task systems. Leung & Merrill [LM80] then derived a feasibility test for periodic task systems by showing that a periodic task system is not feasible iff the deadline algorithm of Liu & Layland and Labetoulle fails to

*This work was supported in part by National Science Foundation Grant No. CCR-8711579.

schedule the system through a predetermined number of time units. Prior to our (this) work, no complete feasibility test for sporadic task systems appears to have been known. Partial and/or related tests have been developed, however; see, e.g., [Mok83, SSL89]. Mok provides a partial reduction from sporadic task systems to periodic task systems, but while schedulability of the resulting periodic task system guarantees schedulability of the sporadic task system, the converse fails to hold. In related work, Sprunt, Sha, & Lehoczky [SSL89] recently provided a reduction from a subclass of sporadic task systems (those in which all deadlines are equal to the respective periods) to periodic task systems with respect to *fixed priority* schedulability, i.e., a fixed priority schedule exists for the sporadic task system iff a fixed priority schedule exists for the periodic task system. (For a discussion on fixed priority schedules see, for example, Leung & Whitehead [LW82].)

The major problem faced in the analysis of sporadic task systems concerns the unpredictability or randomness of the task requests. For periodic task systems the set of task requests to be encountered is known *a priori*. For sporadic task systems, however, there is no *a priori* knowledge about which set of task requests will be encountered.

The ultimate goal regarding sporadic task systems is to find an off-line algorithm that will mechanically synthesize on-line algorithms for scheduling the processor. The off-line algorithm would first determine the *feasibility* of a sporadic task system, and if feasible construct a suitable on-line algorithm for scheduling it. The resulting on-line algorithm must be able to iteratively schedule the processor when presented with any set of task requests that satisfies the separation constraints. Now the deadline algorithm of Dertouzos [Der74] successfully schedules the processor when presented with any set of task requests for which the processor can be scheduled — regardless of whether the separation constraints hold. The deadline algorithm then constitutes a suitable on-line scheduling algorithm whenever one exists. If we could couple the deadline algorithm with a polynomial time or even a pseudo-polynomial time feasibility test we would have made significant progress toward the realization of a viable algorithm for mechanical synthesis.

In this paper, we derive necessary and sufficient conditions for a sporadic task system to be feasible. We first show that a sporadic task system is feasible iff it can be scheduled with respect to a *particular* set of task requests. (This allows us to ignore the random nature of task requests in our analysis.) Techniques similar to those employed by Leung & Merrill [LM80] and Baruah, Howell, & Rosier [BHR90] are then used in deriving our necessary and sufficient conditions. Our conditions immediately yield an exponential time feasibility test for sporadic task systems — the first such test, as far as we know — and suffice for showing that feasibility testing with respect to sporadic task systems belongs to co-**NP**. At this time, we are not able to determine whether feasibility testing is co-**NP**-complete, nor are we able to provide a feasibility test that runs in polynomial or even pseudo-polynomial time. Our test, however, runs efficiently in pseudo-polynomial time for a very large percentage of sporadic task systems with integer parameters. While its worst-case running time is $O(P + \max_{1 \leq i \leq n}\{d_i\})$, where P is the least common multiple (lcm) of $p_1, \ldots, p_n$, it runs in time $O(n \cdot \max_{1 \leq i \leq n}\{p_i - d_i\})$ with a small constant of proportionality for all instances τ where $0.99 < \sum_{i=1}^{n} e_i/p_i \leq 1$ fails to hold.

The remainder of this paper is organized as follows. Section 2 contains material concerning necessary and sufficient conditions for a sporadic task systems to be feasible. The resulting algorithm is presented in Section 3. Although our results are shown with respect to a discrete model of time where the inputs and preemption boundaries are constrained to be integers, the results generalize quite readily to a continuous model of time without integer constraints. We discuss this in Section 4. We also discuss extensions of our results to hybrid systems consisting of some sporadic and some periodic tasks.

2 Feasibility with respect to sporadic task systems

In this section, we derive necessary and sufficient conditions for a sporadic task system to be feasible. Our techniques are similar to those employed by Leung & Merrill [LM80] and Baruah, Howell, & Rosier [BHR90] where it is concluded that feasibility testing with respect to periodic task systems is co-**NP**-complete in the strong sense. The conditions we derive here suffice for showing feasibility testing with respect to sporadic task systems to be in co-**NP**. We are not able to establish a similar **NP**-hardness result, however. Neither are we able to provide a polynomial or pseudo-polynomial time feasibility test. We do, though, in the next section illustrate a feasibility test that runs in pseudo-polynomial time for a very large percentage of sporadic task systems.

Let $\tau = \{T_1, \ldots, T_n\}$ be a sporadic task system where $T_i = (e_i, d_i, p_i)$ with $e_i \leq d_i$ and $e_i \leq p_i$, $1 \leq i \leq$

n. Let $P = \mathrm{lcm}\{p_1, \ldots, p_n\}$. We represent a request by T_i at time t_0 by the pair (i, t_0), $1 \leq i \leq n$, $t_0 \geq 0$. In the context of τ we call the request (i, t_0) a *τ-request.* Recall that (i, t_0) requires that the processor be allocated to T_i for e_i time units during the interval $[t, t + d_i)$. A set of τ-requests R is schedulable iff it is possible for the processor to meet the requirements of all τ-requests in R. A set of τ-requests is *legal* iff whenever R contains both (i, t_1) and (i, t_2) we have $|t_1 - t_2| \geq p_i$, i.e., the separation constraints are satisfied. τ is then feasible iff every legal set of τ-requests is schedulable.

An online scheduling algorithm σ is an iterative algorithm which when presented with a set of τ-requests R (τ-request (i, t_0) in R is presented to σ at time t_0) uniquely determines at each instant of time $t = 0, 1, 2, \ldots$ which *active* τ-request of R — if any — is to be allocated the processor. The τ-request (i, t_0) in R is said to be *active* at time t (with respect to σ) iff $t_0 \leq t < t_0 + d_i$ and σ did not allocate the processor to (i, t_0) for e_i time units in the interval $[t_0, t)$. σ reports failure — but does not terminate — at time t if there is a τ-request (i, t_0) in R such that $t_0 + d_i = t$ and σ did not allocate the processor to (i, t_0) for e_i time units in the interval $[t_0, t)$. σ can only successfully schedule τ if it never reports failure when presented with a legal set of τ-requests. Hence, σ can be expressed as a function of R and t:

$$(\sigma.R).(t) = \begin{cases} (i, t_0), \textit{failure} & \text{meaning that } \sigma \text{ at time } t \text{ allocates the processor to } \tau\text{-request } (i, t_0) \text{ and reports failure.} \\ (0, 0), \textit{failure} & \text{meaning that } \sigma \text{ at time } t \text{ leaves the processor idle and reports failure.} \\ (i, t_0) & \text{meaning that } \sigma \text{ at time } t \text{ allocates the processor to } \tau\text{-request } (i, t_0) \text{ and does not report failure.} \\ (0, 0) & \text{meaning that } \sigma \text{ at time } t \text{ leaves the processor idle and does not report failure.} \end{cases}$$

In what follows we make use of the fact that the deadline algorithm of Dertouzos [Der74] is optimal for sporadic task systems. (See also [LL73, Lab74].) *For the remainder of this paper let σ denote the deadline algorithm.* Given a set of τ-requests R, σ allocates the processor at time t to the active τ-request (if there is one) whose deadline is nearest. Ties can be broken in an arbitrary fashion without affecting whether or when $(\sigma.R)$ reports failure. Hence, without loss of generality we assume σ chooses active τ-request (i, t_1) over (j, t_2) whenever "$t_1 + d_i < t_2 + d_j$" or "$t_1 + d_i = t_2 + d_j$ and $i < j$."

Lemma 1 (Der74) *σ is optimal for sporadic task systems, i.e., given a set of τ-requests R, $(\sigma.R)$ will construct a schedule for R if one exists; otherwise σ at some time will report failure. Thus τ is feasible iff $(\sigma.R)$ constructs a schedule for every legal set of τ-requests R.*

We now investigate properties of τ when τ is not feasible. Our exposition requires the use of the following lemma. Let R be a set of τ-requests, t a non-negative integer, and $h_R(t) = \sum_{(i,t_0) \in R \wedge t_0 + d_i \leq t} e_i$.

Lemma 2 *If $h_R(t) > t$ then*

- *$(\sigma.R)$ will report failure at or before t.*
- *If R is legal τ is not feasible.*

Proof : The number of time units needed in $[0, t)$ to satisfy all τ-requests (i, t_0) in R with $t_0 + d_i \leq t$ is strictly greater than t — the number of time-units in $[0, t)$. Surely then $(\sigma.R)$ will report failure on or before t. If R is legal then Lemma 1 guarantees that τ is not feasible. □

Suppose now that τ is not feasible. Let $t_f \geq 0$ be the earliest time σ reports failure with respect to any legal set of τ-requests. Let R_0 be a legal set of τ-requests such that $(\sigma.R_0)$ reports failure for the first time at time t_f. Consider the finite legal set of τ-requests $R_1 = R_0 - \{(i, t_0) | t_0 + d_i > t_f\}$. Consider $(\sigma.R_1)$. Recall that $(\sigma.R_1)$ mimics $(\sigma.R_0)$ with respect to all tie breaking choices. Over the interval $[0, t_f)$ $(\sigma.R_0)$ never assigns the processor to a τ-request from $R_0 - R_1$ at a time when a request from R_1 is active. Hence, over $[0, t_f)$, $(\sigma.R_0)$ and $(\sigma.R_1)$ make identical assignments with respect to the τ-requests in R_1. Hence, $(\sigma.R_1)$ too reports failure for the first time at time t_f.

We now show by contradiction that $(\sigma.R_1)$ never idles the processor over $[0, t_f)$. Suppose at time t_b, $0 \leq t_b < t_f$, $(\sigma.R_1)$ idles the processor, i.e., $(\sigma.R_1).(t_b) = (0, 0)$. Suppose further that t_b is the largest such integer. Let $R_2 = \{$All τ-requests in R_1 that were not assigned the processor by $(\sigma.R_1)$ in $[0, t_b)\}$. Note that R_2 is legal. Note also that $R_2 \cap \{(i, t_0) | t \leq t_b\} = \phi$. Consider $(\sigma.R_2)$. Now with respect to $(\sigma.R_1)$ no τ-requests are active at time t_b. Thus, over $[t_b, t_f)$, $(\sigma.R_1)$ can only assign the processor to τ-requests in R_2. Hence, $(\sigma.R_2)$ will mimic $(\sigma.R_1)$ over $[t_b, t_f)$, and $(\sigma.R_2)$ will report failure for the first time at t_f. Note now that $(\sigma.R_1)$ and thus $(\sigma.R_2)$ never idles the processor over $[t_b, t_f)$. Lastly consider the finite legal set of τ-requests $R_3 = \{(i, t_0-(t_b+1)) \mid (i, t_0) \in R_2\}$. Then $(\sigma.R_3)$ over $[0, t_f - (t_b+1))$ is identical to $(\sigma.R_2)$ over $[t_b+1, t_f)$, because $(\sigma.R_3)$ over $[0, t_f - (t_b+1))$ mimics the tie breaking choices of $(\sigma.R_2)$ over $[t_b+1, t_f)$. Hence, $(\sigma.R_3)$ reports failure for the first time at $t_f - (t_b+1)$ — a contradiction given our choice for t_f.

Now let us review the properties of R_1:

- Each $(i, t_0) \in R_1$ has $t_0 + d_i \le t_f$,
- $(\sigma.R_1)$ never idles the processor over $[0, t_f)$, and
- $(\sigma.R_1)$ reports failure exactly once — at time t_f.

From these properties we immediately obtain that $\sum_{(i,t_0)\in R_1 \wedge t_0+d_i \le t_f} e_i > t_f$. Hence, from Lemma 2 we see that R_1 is a certificate attesting to the fact that τ is not feasible. Now suppose for some $(i, t_0) \in R_1$ we have:

- $t_0 = q \cdot p_i + r$, $q \ge 0$, $0 < r < t_0$, i.e., $t_0 \equiv r \pmod{p_i}$, $r \ne 0$, and
- each $(i, t') \in R_1$ where $t' < t_0$ is such that $t' \equiv 0 \pmod{p_i}$.

Then we can easily see that a schedule cannot be constructed for $R_4 = R_1 - \{(i, t_0)\} \cup \{(i, q \cdot p_i)\}$ either, since:

- $\sum_{(i,t_0)\in R_4 \wedge t_0+d_i \le t_f} e_i = \sum_{(i,t_0)\in R_1 \wedge t_0+d_i \le t_f} e_i > t_f$.

This implies that from R_1 one can iteratively construct a finite legal set of τ-requests R' such that:

- $R' \subseteq \cup_{i=1}^{n} \cup_{k\ge 0} \{(i, k \cdot p_i)\}$, and
- $\sum_{(i,t_0)\in R' \wedge t_0+d_i \le t_f} e_i > t_f$.

Letting $R = \cup_{i=1}^{n} \cup_{k\ge 0} \{(i, k \cdot p_i)\}$, we then have $\sum_{(i,t_0)\in R \wedge t_0+d_i \le t_f} e_i > t_f$. Hence we now have that R is a certificate attesting to the fact that τ is not feasible.

For the remainder of this section let $\mathcal{R} = \cup_{i=1}^{n} \cup_{k\ge 0} \{(i, k \cdot p_i)\}$. From Lemma 2 and the discussion preceding this paragraph we obtain:

Lemma 3 *τ is not feasible iff there exists an integer $t \ge 0$ such that $h_{\mathcal{R}}(t) > t$. Furthermore, the minimum t satisfying $h_{\mathcal{R}}(t) > t$ is equal to t_f — the earliest time the deadline algorithm can report failure.* □

Recall that $h_{\mathcal{R}}(t) = \sum_{(i,t_0)\in\mathcal{R} \wedge t_0+d_i \le t} e_i$. In what follows, we need a more useful characterization or formula for $h_{\mathcal{R}}(t)$. We now derive such a formula. Clearly, $h_{\mathcal{R}}(t)$ can be written as $\sum_{i=1}^{n} e_i \cdot c(i,t)$, where $c(i,t)$ represents the number of requests (i, t_0) in $\mathcal{R}$ satisfying $t_0 + d_i \le t$. Given our particular $\mathcal{R}$, $c(i,t)$ will be the largest non-negative integer $k+1$ such that k satisfies $k \cdot p_i + d_i \le t$, if one exists; otherwise $c(i,t)$ will equal zero. Since the largest integer satisfying $k \cdot p_i + d_i \le t$ is $\lfloor \frac{t-d_i}{p_i} \rfloor$ we get $c(i,t) = \max\{0, \lfloor \frac{t-d_i}{p_i} \rfloor + 1\}$. We therefore have $h_{\mathcal{R}}(t) = \sum_{i=1}^{n} e_i \cdot \max\{0, \lfloor \frac{t-d_i}{p_i} \rfloor + 1\}$.

Lemma 3 provides a necessary and sufficient condition for τ to be feasible. Its utilitarian value, however, is limited since it provides no *a priori* upper bound on the value of a t such that $h_{\mathcal{R}}(t) > t$, other than t_f. The next three lemmas address this shortcoming and lead to a necessary and sufficient condition that in this respect is more useful.

Lemma 4 *Suppose $\sum_{i=1}^{n} e_i/p_i > 1$. Then τ is not feasible.*

Proof : Suppose $\sum_{i=1}^{n} e_i/p_i > 1$. We will show that there exists a t such that $h_{\mathcal{R}}(t) > t$. Lemma 4 then follows from Lemma 3. Choose $t > \max_{1\le i\le n}\{d_i\}$ such that P divides t and $t > -\sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor / (\sum_{i=1}^{n} \frac{e_i}{p_i} - 1)$. Then:

$h_{\mathcal{R}}(t)$

$= \{\text{By definition}\}$

$$\sum_{i=1}^{n} e_i \cdot \max\{0, \lfloor \frac{t-d_i}{p_i} \rfloor + 1\}$$

$= \{t \ge \max_{1\le i\le n}\{d_i\} \Rightarrow \lfloor \frac{t-d_i}{p_i} \rfloor \ge 0\}$

$$\sum_{i=1}^{n} e_i \cdot (\lfloor \frac{t-d_i}{p_i} \rfloor + 1)$$

$= \{\text{ Pulling the 1 inside the floor}\}$

$$\sum_{i=1}^{n} e_i \cdot \lfloor \frac{t+p_i-d_i}{p_i} \rfloor$$

= {Since $p_i|t$, we pull t/p_i from within the floor function, and rearrange terms}

$$t \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} + \sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor$$

= {rearranging terms}

$$t \cdot (1 + \sum_{i=1}^{n} \frac{e_i}{p_i} - 1) + \sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor$$

= {rearranging terms}

$$t + t \cdot (\sum_{i=1}^{n} \frac{e_i}{p_i} - 1) + \sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor$$

> {Substituting $-\sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor /$ $(\sum_{i=1}^{n} \frac{e_i}{p_i} - 1)$ for the latter t}

$$t + \frac{(-\sum_{1=1}^{n} e_i \lfloor \frac{p_i - d_i}{p_i} \rfloor)}{(\sum_{1=1}^{n} \frac{e_i}{p_i} - 1)} \cdot$$

$$(\sum_{1=1}^{n} \frac{e_i}{p_i} - 1) + \sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor$$

= {cancelling the common factor $(\sum_{i=1}^{n} \frac{e_i}{p_i} - 1)$}

$$t - \sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor + \sum_{i=1}^{n} e_i \cdot \lfloor \frac{p_i - d_i}{p_i} \rfloor$$

= {Cancelling terms}

$$t.$$

Hence, $h_{\mathcal{R}}(t) > t$. □

Lemma 5 *Suppose* $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ *and* $t \geq \max_{1 \leq i \leq n}\{d_i\}$. *Then* $h_{\mathcal{R}}(t + P) > t + P$ *implies* $h_{\mathcal{R}}(t) > t$.

Proof : Suppose $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$, $t \geq \max_{1 \leq i \leq n}\{d_i\}$, and $h_{\mathcal{R}}(t + P) > t + P$. Then:

$h_{\mathcal{R}}(t) + P$

= {By definition of $h_{\mathcal{R}}(t)$, and the assumption that $t \geq \max_{1 \leq i \leq n}\{d_i\}$}

$$\sum_{i=1}^{n} e_i \cdot (\lfloor \frac{t - d_i}{p_i} \rfloor + 1) + P$$

≥ {since $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$}

$$\sum_{i=1}^{n} e_i \cdot (\lfloor \frac{t - d_i}{p_i} \rfloor + 1) + P \sum_{i=1}^{n} \frac{e_i}{p_i}$$

= {rearranging the second term}

$$\sum_{i=1}^{n} e_i \cdot (\lfloor \frac{t - d_i}{p_i} \rfloor + 1) + \sum_{i=1}^{n} e_i \cdot \frac{P}{p_i}$$

={Rearranging terms}

$$\sum_{i=1}^{n} e_i \cdot (\lfloor \frac{t - d_i}{p_i} \rfloor + \frac{P}{p_i} + 1)$$

= {Move P inside the floor, since $p_i|P$}

$$\sum_{i=1}^{n} e_i \cdot (\lfloor \frac{t + P - d_i}{p_i} \rfloor + 1)$$

= {By definition}

$$h_{\mathcal{R}}(t + P)$$

> {By assumption}

$$t + P$$

Now $h_{\mathcal{R}}(t) + P > t + P$ immediately yields the desired conclusion — $h_{\mathcal{R}}(t) > t$. □

Corollary 1 *Suppose* τ *is not feasible and* $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$. *Then there exists a* $t < P + \max_{1 \leq i \leq n}\{d_i\}$ *such that* $h_{\mathcal{R}}(t) > t$.

Lemma 6 *Suppose* τ *is not feasible and* $\sum_{i=1}^{n} \frac{e_i}{p_i} < 1$. *Then* $h_{\mathcal{R}}(t) > t$ *implies* $t < \max_{1 \leq i \leq n}\{d_i\}$ *or* $t < \max_{1 \leq i \leq n}\{p_i - d_i\} \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} / (1 - \sum_{i=1}^{n} \frac{e_i}{p_i})$.

Proof : Assume that τ is not feasible, $\sum_{i=1}^{n} \frac{e_i}{p_i} < 1$, $h_{\mathcal{R}}(t) > t$, and $t \geq \max_{1 \leq i \leq n}\{d_i\}$. Then our proof obligation is to show that t must be less than $\max_{1 \leq i \leq n}\{p_i - d_i\} \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} / (1 - \sum_{i=1}^{n} \frac{e_i}{p_i})$.

$h_{\mathcal{R}}(t)$

= {By definition of $h_{\mathcal{R}}(t)$}

$$\sum_{i=1}^{n} e_i \cdot \max\{0, \lfloor \frac{t - d_i}{p_i} \rfloor + 1\}$$

$\leq$ {Eliminating the floor function, and rearranging terms}

$$\sum_{i=1}^{n} e_i \cdot (\frac{t+p_i-d_i}{p_i})$$

= {Rearranging terms }

$$t \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} + \sum_{i=1}^{n} \frac{e_i}{p_i}(p_i - d_i)$$

$\leq$ {since $\max_{1\leq i \leq n}\{p_i - d_i\} \geq (p_j - d_j)$ for all j, $1 \leq j \leq n$}

$$t \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} + \max_{1\leq i\leq n}\{p_i - d_i\} \sum_{i=1}^{n} \frac{e_i}{p_i}$$

Substituting t for $h_{\mathcal{R}}(t)$, and from the assumption that $h_{\mathcal{R}}(t) > t$, we obtain:

$$t < t \cdot \sum_{i=1}^{n} \frac{e_i}{p_i} + \max_{1\leq i\leq n}\{p_i - d_i\} \cdot \sum_{i=1}^{n} \frac{e_i}{p_i},$$

$\equiv$ { Rearranging terms, and factoring out t}

$$t(1 - \sum_{i=1}^{n} \frac{e_i}{p_i}) < \max_{1\leq i\leq n}\{p_i - d_i\} \cdot \sum_{i=1}^{n} \frac{e_i}{p_i}$$

$\equiv$ {dividing both sides by the positive quantity $(1 - \sum_{i=1}^{n} \frac{e_i}{p_i})$}

$$t < \max_{1\leq i\leq n}\{p_i - d_i\} \cdot \frac{\sum_{i=1}^{n} \frac{e_i}{p_i}}{(1 - \sum_{i=1}^{n} \frac{e_i}{p_i})}.$$

□

Lemmas 3-6 and Corollary 1 yield:

Theorem 1 *Let $\sum_{i=1}^{n} \frac{e_i}{p_i} = c$. τ is not feasible iff either:*

(1) *$c > 1$, or*

(2) *there exists an integer t, $t < \min\{P + \max_{1\leq i\leq n}\{d_i\}, \frac{c}{1-c}\max_{1\leq i\leq n}\{p_i - d_i\}\}$ such that $h_{\mathcal{R}}(t) > t$.*

Corollary 2 *Feasibility testing with respect to sporadic task systems is in co-*NP.

Proof : The corollary follows directly from Theorem 1 once one observes that:

- an integer $t < P + \max_{1\leq i\leq n}\{d_i\}$ can be "guessed" in nondeterministic polynomial time, and
- checking whether $h_{\mathcal{R}}(t) > t$ or whether $\sum_{i=1}^{n} \frac{e_i}{p_i} > 1$ can be accomplished in deterministic polynomial time. □

3 An algorithm for feasibility testing with respect to sporadic task systems

In the previous section we derived necessary and sufficient conditions for a sporadic task system to be feasible. In this section, we present a feasibility testing algorithm for sporadic task systems based on these conditions. In the worst case our algorithm runs in exponential time; however, for a very large percentage of sporadic task systems the algorithm runs in pseudo-polynomial time. This "speed-up" with respect to this large percentage of systems represents an exponential improvement over the algorithm's worst case behavior.

Again let $\tau = \{T_1, \ldots, T_n\}$ where $T_i = (e_i, d_i.p_i)$ with $e_i \leq d_i$ and $e_i \leq p_i$, $1 \leq i \leq n$, be a sporadic task system. Let $P = \text{lcm}\{p_1, \ldots, p_n\}$. Let $\mathcal{R} = \cup_{i=1}^{n} \cup_{k\geq 0} \{(i, k \cdot p_i)\}$.

An obvious feasibility testing algorithm might simulate the deadline algorithm over $\mathcal{R}$ until it either reports failure or until it is guaranteed to never report failure. Our algorithm "essentially" does this without the the mess and cleanup involved in performing an actual simulation. Our algorithm — based on the observations stated in Theorem 1 — iteratively searches for an integer $t > 0$ such that $h_{\mathcal{R}}(t) > t$. The "simulation" here is done implicitly via the computation of $h_{\mathcal{R}}$. Recall that Lemma 3 guarantees finding a t such that $h_{\mathcal{R}}(t) > t$ occurs after iterating through exactly t_f values for t, where t_f is the earliest time the deadline algorithm can report failure.

Before presenting the algorithm we prove a technical result that yields a small improvement over the algorithm suggested by Theorem 1.

Lemma 7 *Suppose $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ and $d_i \geq p_i$, $1 \leq i \leq n$. Then τ is feasible.*

Proof : Suppose $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ and $d_i \geq p_i$, $1 \leq i \leq n$. We show that $h_{\mathcal{R}}(t) \leq t$ for all integers $t \geq 0$. The lemma then follows directly from Lemma 3. Now :

$h_{\mathcal{R}}(t)$

$= \{\text{By definition}\}$

$$\sum_{i=1}^{n} e_i \cdot \max\{0, \lfloor \frac{t-d_i}{p_i} \rfloor + 1\}$$

$\leq$ { By assumption, $d_i \geq p_i$ for all i, $1 \leq i \leq n$ }

$$\sum_{i=1}^{n} e_i \cdot \lfloor \frac{t}{p_i} \rfloor$$

$\leq$ {Getting rid of the "floor."}

$$t \cdot \sum_{i=1}^{n} \frac{e_i}{p_i}$$

$\leq$ { By assumption, $\sum_{i=1}^{n} e_i/p_i \leq 1$}

$$t$$

□

We now present our algorithm for feasibility testing with respect to sporadic task systems.

```
input($\tau$);
/* Recall that $\tau = \{T_1, \ldots, T_n\}$,
$e_i \leq d_i$, $e_i \leq p_i$, for $1 \leq i \leq n$ */
$c := \sum_{i=1}^{n} \frac{e_i}{p_i}$;
if "$c > 1$" then return("not feasible");
/* Correctness guaranteed by Lemma 4 */
if "$\wedge_{i=1}^{n}(d_i \geq p_i)$" then return("feasible");
/* Correctness guaranteed by Lemma 7 */
$P := \text{lcm}\{p_1, \ldots, p_n\}$;
$D := \max_{1 \leq i \leq n}\{d_i\}$;
$M := P + D$;
if "$c = 1$" then $T := M$
  else $T := \min\{M, \frac{c}{1-c} \cdot \max_{1 \leq i \leq n}\{p_i - d_i\}\}$;
$H := 0$;
/* $H$ will contain the value of $h_{\mathcal{R}}(t)$, and,
by definition, $h_{\mathcal{R}}(0) = 0$ */
for $t := 1$ to $T$ do
  for $i := 1$ to $n$ do
    if "$t \geq d_i$" and "$t \equiv d_i \pmod{p_i}$"
      then $H := H + e_i$;
  endfor
/* $H$ now equals $h_{\mathcal{R}}(t)$ */
if "$H > t$" then return("not feasible");
/* Correctness guaranteed by Lemma 3 */
endfor
return("feasible");
/* Correctness guaranteed by Theorem 1 */
end.
```

Theorem 2

(1) *If "$\sum_{i=1}^{n} \frac{e_i}{p_i} > 1$" holds then our algorithm runs in linear time.*

(2) *If "$\wedge_{i=1}^{n} d_i \geq p_i$" holds then our algorithm runs in linear time.*

(3) *Let c be a fixed constant that is strictly less than one — for example, say 0.99. Then our algorithm runs in $O(n \cdot \max_{1 \leq i \leq n}\{p_i - d_i\})$ time.*

(4) *If none of the above cases hold then the algorithm runs in exponential time — $O(n \cdot (P + \max_{1 \leq i \leq n}\{d_i\}))$ time.* □

Now, "$\sum_{i=1}^{n} \frac{e_i}{p_i} > 1$" is not likely to hold for the majority of inputs. Neither is "$\wedge_{i=1}^{n}(d_i \geq p_i)$." Hence, the pruning accomplished by (1) and (2) affects only slightly the algorithm's expected run time. However, we think it is reasonable to expect that the vast majority of inputs will satisfy "$0 < \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 0.99$," and in this case the algorithm runs in pseudo-polynomial time — $O(n \cdot \max_{1 \leq i \leq n}\{p_i - d_i\})$ — with a small constant of proportionality — 99. Hence:

Corollary 3 *Feasibility testing can be accomplished in $O(n \cdot \max_{1 \leq i \leq n}\{p_i - d_i\})$ deterministic time, when $0.99 < \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$ fails to hold.* □

One final comment : Note that $O(n \cdot \max_{1 \leq i \leq n}\{p_i - d_i\})$ is potentially exponentially less than $O(P + \max_{1 \leq i \leq n}\{d_i\})$ as P can be as large as $\prod_{i=1}^{n} p_i$. Hence, the "speed-up" for sporadic task systems where $0 < \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 0.99$ is potentially exponential with respect to the algorithm's worst-case behavior.

4 Conclusions

Highlights concerning results presented in this paper are

- Necessary and sufficient conditions are derived for a sporadic task system to be feasible.
- Feasibility testing with respect to sporadic task systems is shown to be in co-**NP**.
- A feasibility testing algorithm for sporadic task systems is provided that runs efficiently in pseudo-polynomial time when $0 < \sum_{i=1}^{n} \frac{e_i}{p_i} \leq 0.99$ or $1 < \sum_{i=1}^{n} \frac{e_i}{p_i}$, and all inputs are integers.

The results presented here are similar to the results of Baruah, Howell, & Rosier [BHR90] concerning synchronous periodic task systems.

A **periodic task** T is characterized by four non-negative integers — s the *start time*, e the *execution time*, d the deadline (relative to a request time), and p the *period* (the time between T's successive requests). For a periodic task it is required that $0 < e \leq d \leq p$. (See [LL73, LM80, LM81, LW82, Mok83, BHR90].) Task T's i'th, $i = 1, 2, \ldots$, request occurs at time $s + i \cdot p$, and requires that the processor be allocated to T for e time units in the interval $[s + i \cdot p,\ s + i \cdot p + d)$. A periodic task system τ is a collection of periodic tasks $\{T_1, \ldots, T_n\}$, with $T_i = (s_i, e_i, d_i, p_i)$, $1 \leq i \leq n$. Liu & Layland [LL73] published the first significant results regarding periodic task systems. When $d_i = p_i$, $1 \leq i \leq n$, they show τ is feasible on one processor iff $\sum_{i=1}^{n} e_i/p_i \leq 1$. See also Coffman [Cof76]. In addition, Liu & Layland show that the deadline algorithm is optimal for periodic task systems on one processor. The deadline algorithm remains optimal for periodic task systems when the $d_i = p_i$ restriction is removed although feasibility testing then becomes much harder. Leung and Merrill [LM80] showed that feasibility testing in this case is co-**NP**-hard. Baruah, Howell, & Rosier [BHR90] subsequently refined this to co-**NP**-complete in the strong sense.

Now, τ is *synchronous* iff $s_i = 0$, $1 \leq i \leq n$. In [BHR90] one can find a feasibility test for synchronous periodic task systems that is similar to the algorithm presented here for sporadic task systems. Similar techniques apply here because the synchronous periodic task system is feasible iff the deadline algorithm σ never reports failure with respect to $\mathcal{R} = \cup_{i=1}^{n} \cup_{k \geq 0} \{(i, k \cdot p_i)\}$. In fact, many of the results shown here can be viewed as extensions of results derived in [BHR90] — extensions for synchronous periodic task systems where "$e_i \leq d_i \leq p_i$" need not be satisfied but "$e_i \leq d_i$ and $e_i \leq p_i$" must. Consequently, all of the results presented in this paper hold for hybrid task systems consisting of some sporadic tasks and some synchronous periodic tasks (periodic task $T_i = (s_i, e_i, d_i, p_i)$ is synchronous iff $s_i = 0$). When all the periodic tasks are not synchronous, the feasibility problem is clearly co-**NP**-hard in the strong sense, since the feasibility problem for periodic task systems is co-**NP**-complete in the strong sense [BHR90]. It can be shown that the feasibility problem for general hybrid task systems is also co-**NP**-complete in the strong sense — details will be provided in an extended version of this paper, currently under preparation.

While the results in Sections 2 and 3, and those mentioned above, were derived for task systems with integer parameters, most extend rather naturally to systems with real-valued parameters. The definition of $h_R(t)$ (Section 2) may be extended such that $h_R(t) = \sum_{(i,t_0) \in R \wedge t_0 + d_i \leq t} e_i$ for all *non-negative real numbers* t. With this extended definition, Lemmas 1-7, Corollaries 1 and 2, and Theorem 1 continue to hold, except that t may now be any non-negative real number (i.e., it need no longer be an integer). The algorithm presented in Section 3 makes use of the fact that deadlines may occur only at integer boundaries; for systems with real-valued parameters, this is no longer true. The algorithm may be extended for inputs which are rational numbers by multiplying all inputs by a common denominator; however, when such an extension is made to a pseudo-polynomial-time algorithm, the resulting algorithm is potentially exponential in n. To avoid this problem in extending Theorem 2(3), we must first modify the algorithm to compute the summation for only those times t at which a deadline occurs. The resulting algorithm not only works for noninteger inputs, it operates in time $O(n^2 \max\{p_i - d_i\} / \min\{p_i\})$, which is slightly better than the original algorithm on integer inputs, as long as every period is at least n. For Theorem 2(4), the extended algorithm operates in time $O(n^2 \cdot (P + \max\{d_i\}) / \min\{p_i\})$.

References

[BHR90] S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems*, 1990. To appear.

[Cof76] E. Coffman, Jr. et. al. *Computer and Job-Shop Scheduling Theory.* Wiley, 1976.

[Der74] M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[HL88] K. Hong and J. Leung. On-line scheduling of real-time tasks. In *Proceedings of the Real-Time Systems Symposium*, pages 244–250, Huntsville, Alabama, December 1988. IEEE.

[JAM90] K. Jeffay, R. Anderson, and C. Martel. On optimal, non-preemptive scheduling of periodic and sporadic tasks. Technical Report TR-90-019, Department of Computer Science, The University of North Carolina at Chapel Hill, April 1990.

[Lab74] J. Labetoulle. Some theorems on real-time scheduling. In E. Gelenbe and R. Mahl, editors, *Computer Architecture and Networks*, pages 285–293. North-Holland, 1974.

[LL73] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, 1973.

[LM80] J. Leung and M. Merrill. A note on the preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11:115–118, 1980.

[LM81] E. Lawler and M. Mertel. Scheduling periodically occurring tasks on multiple processors. *Information Processing Letters*, 12:9–12, 1981.

[LSS87] J. Lehoczky, L. Sha, and J. Stronider. Enhanced aperiodic responsiveness in hard real-time environments. In *Proceedings of the Real-Time Systems Symposium*, pages 261–270, San Jose, CA, December 1987. IEEE.

[LW82] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[Mok83] A. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, MIT Laboratory for Computer Science, May 1983.

[SLS88] B. Sprunt, J. Lehoczky, and L. Sha. Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. In *Proceedings of the Real-Time Systems Symposium*, pages 251–258, Huntsville, Alabama, December 1988. IEEE.

[SSL89] B. Sprunt, L. Sha, and J. Lehoczky. Scheduling sporadic and aperiodic events in a hard real-time system. Technical Report ESD-TR-89-19, Carnegie Mellon University, 1989.