Linear Time On-Line Feasibility Testing Algorithms for Fixed-Priority, Hard Real-Time Systems

Michael E. Thomadakis
Computer Science Department
Texas A&M University
College Station
TX 77843–3112

`miket@cs.tamu.edu`

Abstract

In this paper[1] we develop linear time, on-line feasibility testing algorithms for the guaranteed scheduling of firm aperiodic tasks in fixed-priority real-time systems. Firm aperiodic, unlike critical periodic tasks which are guaranteed off-line, can be guaranteed dynamically only if they pass on-line feasibility tests. In this paper we derive feasibility tests requiring a one time $\Theta(n)$ computation time, where n is the number of periodic tasks. Our tests draw upon the *Workload-Matrix* method, an innovative technique which determines the idle capacity within arbitrary intervals $[t_1, t_2)$ of a periodic schedule *exactly*. To the best of our knowledge, this is the first algorithm reported in the literature, which can perform on-line schedulability testing in linear time. Previous state-of-the-art on-line schedulability methods are based on either static or dynamic slack stealing, or EDL methods by Silly and Chetto, and they require pseudo-polynomial time to guarantee one single task and they incur continuously overhead for state variable maintenance. The proposed method utilizes the entire spare capacity in the schedule, it maintaines guarantess to the periodic tasks, and experiments show that the computation time for admission control is in the order of a few μ-secs on commodity Unix workstations.

Keywords

Firm-Aperiodic Tasks; Fixed Priority Preemptive Scheduling; FPF/LDL Schedules; Hard Periodic Real-Time Tasks; Idle-Capacity Management; On-Line Feasibility Algorithms; Workload Processes

I. Introduction

In this work we develop linear time on-line feasibility testing algorithms which can be used in the admission control procedure of *firm* aperiodic tasks. We consider fixed-priority scheduling schemes in which periodic real-time tasks with hard deadlines are assigned fixed-priorities by a method, such as, the Deadline Monotonic rule [1], and are guaranteed off-line. Firm tasks generate no utility for the system if they finish after their deadlines. They have unknown parameters and arrive at unpredetermined instances of time, and thus, they cannot be guaranteed off-line. The system affords

[1] Note to the reviewers: we have included an Appendix for ease of reference. We do not plan to include this Appendix in the final manuscript in case it is accepted. We will provide, instead, a Tech-Report accessible at the site of our department with the full version of this paper.

an *on-line guarantee* to each new firm task at the time of its arrival. The guaranteed scheduling of firm tasks involves three logical steps. First, upon the arrival of a new firm aperiodic J_a at time r_a, the system verifies that the schedule of already guaranteed tasks will remain feasible if J_a is admitted. In the second step, if feasibility is verified, the system performs *schedule maintenance*, where it determines the points in time at which the affected tasks will receive the processor. Finally, the system dispatches the tasks according to the modified schedule. The real-time system may admit a new firm task, only if, there is sufficient idle processor capacity to accommodate its execution requirements. The on-line feasibility testing then reduces to a method which determines the idle capacity, denoted as $Z^S[r_a, d_a)$, where d_a is the firm deadline of J_a.

Efficient determination of the exact idle processor capacity in fixed and dynamic priority schedules has been a challenging problem. Presently, state-of-the-art algorithms, such as, Slack Stealing [2], [3] or EDL methods [4], [5], require pseudo-polynomial time complexity in order to test the feasibility of one firm task on-line. Several methods have been devised to compute a lower bound on the idle capacity approximately [6], [7], [8]. However, approximate methods can at times become overly pessimistic and thus, reject several schedulable firm tasks. Both exact and approximate methods incur high run-time overheads for continuous slack variable maintenance, even when there are no firm tasks present.

In this paper we develop algorithms based on exact feasibility conditions, which guarantee and schedule on-line firm aperiodic tasks, in linear time. Our primary concern, besides exactness, is low complexity and execution time overheads. For a system with n periodic tasks, $t_i,\ i = 1, \ldots, n$, we develop a scheme where idle capacity $Z_i^S(t)$ is determined as a deterministic process of time, defined as the total idle time in $[0, t)$ in a schedule S of the first i periodic tasks. This is obtained as $t - W_i^S(t)$, where $W_i^S(t)$ is defined as the total busy time in schedule S in $[0, t)$, for the first i periodic tasks. In the following sections we provide computational procedures for the exact calculation of $W_i^S(t)$ for arbitrary t. A fixed priority schedule of the first i periodic tasks is **legal** if all job instances τ_{ij} of task τ_i are assigned to the processor after they are released at times r_{ij}. The schedule is **feasible** if all instances τ_{ij} always receive the processor for their C_i computation requirements units, in every $[r_{ij}, d_{ij})$, where d_{ij} is the absolute deadline of instance τ_{ij}. In essence, for the same periodic task set $\boldsymbol{\tau}$ infinitely many feasible schedules can be defined, when the task parameters are rational numbers. Different schedules consist of different sequences of busy and idle periods, where periodic tasks are serviced over the busy periods. Firm tasks can only be scheduled in the idle periods of some feasible periodic schedule, otherwise some instance τ_{kl} will miss its deadline. We formally define two specific fixed-priority periodic schedules, called the *Fixed-Priority First* (FPF) and the *Latest-Deadline Last* (LDL). The FPF is a non-idling schedule where instances are dispatched to the processor at their earliest possible time after their release, according to the Highest-Priority Next (HPN) dispatching rule. The LDL is an idling schedule where instances are dispatched at their latest possible time, while they all can meet their deadlines. The FPF and the LDL are the two unique extremes fixed-priority schedules for a task set $\boldsymbol{\tau}$. We prove the *Workload Dominance* theorem by which $W_i^{\mathrm{F}}(t) \geq W_i^{\mathrm{L}}(t)$, where F stands for FPF and L for LDL. When a firm task J_a requests C_a computation units in $[r_a, d_a)$ if $d_a - r_a - (W_n^{\mathrm{F}}(d_a) - W_n^{\mathrm{F}}(r_a)) < C_a$ the system cannot accept J_a under the FPF periodic schedule. By the Workload Dominance theorem if we start dispatching the tasks at time r_a under the LDL schedule, it is possible to shift additional idle capacity within $[r_a, d_a)$ allowing the possibility to guarantee task J_a. In subsequent sections we derive the two extreme schedules formally and develop efficient computational procedures for the $W_i^{\mathrm{F}}(t)$ and $W_i^{\mathrm{L}}(t)$. We note that idling schedules have been known for some time and several methods, such as slack stealing or EDL, attempted to utilize them. However, previously proposed algorithms based on non-idling schedules are all pseudo-polynomial in time complexity, and they incur high run-time overheads, even when no firm tasks are present.

The rest of this paper is organized as follows. Section II derives the two extreme schedules. Section III develops two computational procedures for $W_i^S(t)$. Section IV presents experimental results on the effectiveness and actual execution time efficiency of our idle capacity determination algorithms. Finally, Section V concludes this paper and highlights several scheduling algorithms which utilize the feasibility testing methods. Section B in the Appendix discusses on-line feasibility methods in terms of time complexity and other overheads.

II. Real-Time System Model and Scheduling Framework

A. Task Model and Schedules for Periodic Tasks

Periodic tasks with fixed periods and hard deadlines are typically common in real-time systems [9], [10], [11]. We consider a set $\boldsymbol{\tau}$ of n hard real-time periodic tasks denoted as $\boldsymbol{\tau} = \{\tau_i(C_i, T_i, D_i, r_i);\ i = 1, \ldots, n\}$. For each task τ_i, $C_i > 0$ is the worst case (maximum) execution time, $T_i > 0$ is the request period, D_i is its relative deadline, with $C_i \leq D_i \leq T_i$ and $r_i \geq 0$ is its initiation displacement relative to time origin $t = 0$. Task τ_i generates an infinite sequence of jobs τ_{ij}, $j = 1, 2, \ldots$, in which τ_{ij} is released at time instance $r_{ij} = r_i + (j-1)T_i$ and must complete by time $d_{ij} = r_{ij} + D_i$. The *hyper-period* H of a periodic task set is defined as the least common multiple (LCM) of the request periods T_i, that is $H := \mathrm{LCM}\{T_1, T_2, \ldots, T_n\}$. When all $r_1 = r_2 = \cdots = r_n = a \geq 0$, the task system is called *synchronous*, and, without loss of generality, we assume $a = 0$. When $r_j \neq r_k$, for some j, k, then the system is called *asynchronous*, and we let

$r = \max_{i=1}^{n}\{r_i\}$.

Hard deadline, non-periodic activities are considered as *sporadic* and firm *aperiodic* tasks. Sporadic tasks $J_{si}(C_{si}, T_{si}, D_{si}, r_{si})$, embody activities with a minimum period of occurrence T_{si}. The other sporadic task parameters are the same as those of a periodic one. For the purpose of analysis we consider the sporadic tasks as periodic ones arriving at their maximum frequency, so that we can guarantee them off-line,.

Firm aperiodic tasks, denoted as $J_{ai}(C_{ai}, D_{ai}, r_{ai})$, characterize dynamic activities with parameters that become known at their release times. The system cannot guarantee all firm aperiodic tasks. We consider the aperiodic tasks as a sequence $\boldsymbol{J}_a := \{J_{ai}(C_{ai}, D_{ai}, r_{ai});\ i = 1, 2, \dots\}$, where task J_{ai} arrives at time r_{ai}, requests C_{ai} units of computation and has to complete by time $d_{ai} = r_{ai} + D_{ai}$. The scheduler calculates dynamically the total idle processor capacity $Z(r_{ai}, d_{ai})$, taking into account periodic, sporadic and already admitted hard aperiodic tasks that have to execute within the interval $[r_{ai}, d_{ai}]$. It accepts J_{ai}, *only if*, $C_{ai} \leq Z(r_{ai}, d_{ai})$, otherwise it rejects J_{ai}, immediately.

A *continuous time schedule* $\mathcal{S}$ of a task system $\boldsymbol{\tau}$ is a mapping of time $t \in \mathbb{R}$ (the real numbers) into the index set of the tasks $\{1, 2, \dots, n\}$, and $\{0\}$, symbolically,

$$\mathcal{S} : \mathbb{R} \longrightarrow \{0, 1, 2, \dots, \mathrm{n}\}. \tag{1}$$

In other words, $\mathcal{S}(t)$ is the index of the task assigned to the processor at time t. When $\mathcal{S}(t) = 0$, no task is assigned to the processor and the system is idle. We let $s_i(t)$ be an indicator function for task τ_i, so that $s_i(t) := 1$ if $\mathcal{S}(t) = i$, and 0, otherwise. A scheduling algorithm $A^{\mathcal{S}}$ is a method that assigns a unique task index to $\mathcal{S}(t)$ for each $t \geq 0$. From the schedule definition we exclude instantaneous processor assignments, as uninteresting cases.

B. Deterministic Processes of the System Workload

In the remainder of the paper we will be using the following quantities repeatedly,

$$n_i(t) = \left\lfloor \frac{t - r_i}{T_i} \right\rfloor + 1,$$
$$m_i(t) = (t - r_i) \pmod{T_i}.$$

$n_i(t)$ is the current period number of τ_i at t, and $m_i(t)$ is the corresponding *translated* remainder, for $i = 1, 2, \dots, n$. For any $t \geq r_i$, $t = r_i + (n_i(t) - 1) \cdot T_i + m_i(t)$ and $r_{i,n_i(t)} \leq t < r_{i,n_i(t)+1}$. For a

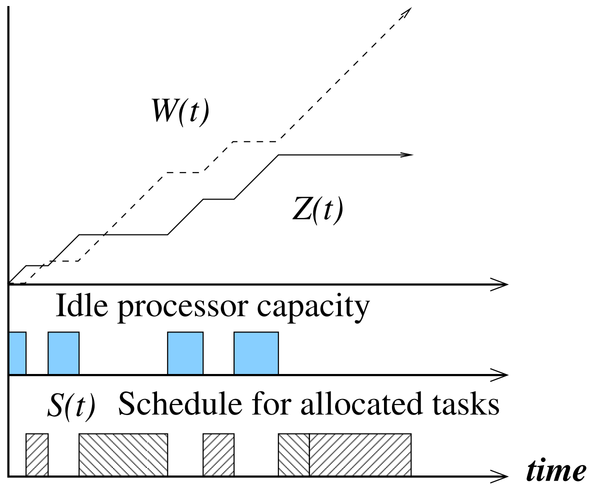


Fig. 1. Illustration of Serviced Workload $W^S(t)$ and Idle Capacity $Z^S(t)$ Processes on Schedule $\mathcal{S}(t)$

given periodic task set $\boldsymbol{\tau}$ we define the *released workload* functions $v_i(t), V_i(t)$ as,

$$v_i(t) := n_i(t) \cdot C_i \quad \text{and} \quad V_i(t) = \sum_{k=1}^{i} v_k(t). \tag{2}$$

$v_i(t), V_i(t)$ is the total amount of work that task τ_i and tasks $\tau_1, \dots, \tau_i$, respectively, require of the system in the time interval $[0, t]$. $V_i(t)$ are right-continuous, step-functions with jump points at the release times of the tasks. $V_i(t)$ have identical increments every H time units (see Subsection III.C.3 in [12] for a proof). For the firm aperiodic tasks $\boldsymbol{J}_{ai}$ we define the released workload $V_a(t)$, as $V_a(t) := \sum_{k=1}^{m} C_{ak},\ r_{am} \leq t < r_{a,m+1}$. The aggregate released workload $\boldsymbol{V}(t)$ of the system in $[0, t]$, is simply $\boldsymbol{V}(t) = V_n(t) + V_a(t),\ t \geq 0$.

Given a schedule $\mathcal{S}$ on $\boldsymbol{\tau}$, we define the *serviced workload* processes $w_i(t), W_i(t)$, that measure the amount of work the system has completed in the time interval $[0, t]$. We let $w_i(t)$ and $W_i(t)$ denote the total time the system has spent processing task τ_i and $\tau_k,\ k = 1, \dots, i$, respectively, in the interval $[0, t]$, defined as

$$w_i(t) := \int_0^t s_i(x)dx \quad \text{and} \quad W_i(t) = \sum_{k=1}^{i} w_k(t). \tag{3}$$

$w_i(t)$ and $W_i(t)$ are piece-wise continuous, linear, non-decreasing functions of t. For each i, when $s_i(t) = 1$, $w_i(t)$ and $W_i(t)$ are increasing with slope at a 45^o, and when $s_i(t) = 0$, $w_i(t)$ stays parallel to the x-axis. $W_i(t)$ is increasing with slope at a 45^o, if any of the tasks $\tau_k,\ k = 1, 2, \dots, i$ is receiving service at t. Note that $W_n(t_2) - W_n(t_1)$ is the total time that the processor is *busy* in the time interval $[t_1, t_2]$.

Given the process $W_i(t)$ we define the *Idle Processor Capacity* processes $Z_i(t)$, as the total amount of time the processor is idle during the interval $[0, t)$, as,

$$Z_i(t) := t - W_i(t),\ i = 1, 2, \ldots, n;\ t \geq 0. \tag{4}$$

Fig. 1 illustrates the serviced $W(t)$ and idle-capacity $Z(t)$ processes on some schedule $\mathcal{S}(t)$ of arbitrary tasks. In a schedule of periodic tasks, $Z_i(t_2) - Z_i(t_1)$ measures the available processor capacity remaining after the jobs of tasks $\tau_1, \tau_2, \ldots, \tau_i$, receive service in $[t_1, t_2)$.

C. Extreme Schedules for Periodic Task Sets

In our discussion we make the assumptions that the task system $\boldsymbol{\tau}$ is *schedulable*, that there is a known fixed priority assignment determined, for instance, by the deadline monotonic (DM) [1] rule, and that the tasks are already indexed according to this priority assignment, with smaller indices implying higher priorities. In the next section, we develop efficient algorithms to compute the serviced workload $W_i(t)$ functions defined on them.

C.1 The Fixed-Priority-First (FPF) Non-Idling Schedule

We consider first the *Fixed-Priority First* (FPF), a schedule which directly reflects the given fixed priority assignment of the periodic tasks. FPF, denoted by $\mathcal{S}^{\mathrm{F}}$, is a *non-idling* schedule, dispatching the jobs τ_{ij} to the processor immediately after they arrive at r_{ij}, according to the usual Highest Priority Next (HPN) rules. FPF models the regular behavior of the system when there are no aperiodic tasks present. Fig. 2 illustrates a schedule by FPF of a simple task set. The top row shows the sequence of job executions and the resulting busy and idle periods. The formal derivation of $\mathcal{S}^{\mathrm{F}}$ first determines the schedule $s_1(t),\ t > 0$ and quantities $W_1(t), Z_1(t), s_{1j}, f_{1j},\ j = 1, 2, \ldots, n_1(H)$ of task τ_1. Subsequently, the algorithm derives the same quantities for the remaining tasks inductively, one task at a time, in the order $\tau_2, \tau_3, \ldots, \tau_n$, with step i utilizing the quantities derived from step $i - 1$.

Specifically, setting $V_0(t) = W_0(t) = 0$, for each $t \geq 0$, and $j = n_i(t)$, let

$$s_i(t) := \begin{cases} 1, & \text{if } i = 1, t \in [r_{ij}, r_{ij} + C_i), \\ 1, & \text{if } i > 1, Z_{i-1}(t) - Z_{i-1}(r_{ij}) < C_i \text{ and } \mathcal{S}_{i-1}(t) = 0, \\ & \text{for } t \in [r_{ij}, d_{ij}), \\ 0, & \text{otherwise;} \end{cases} \tag{5}$$

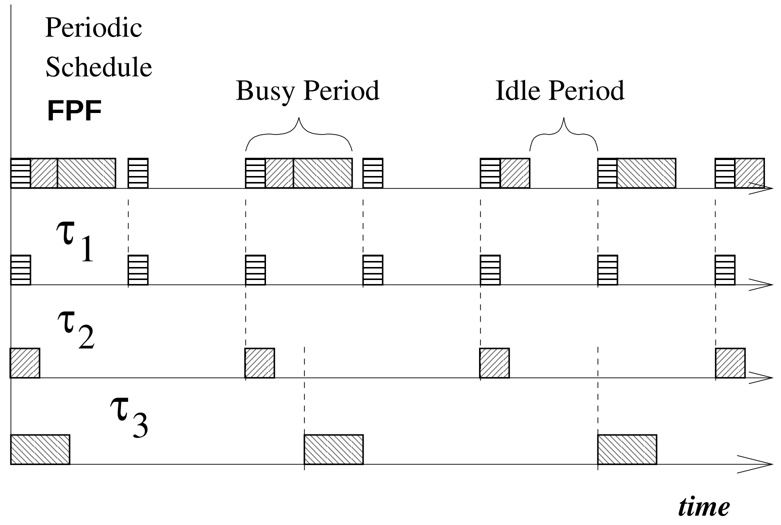


Fig. 2. The Non-Idling Fixed-Priority-First Schedule of a Random Task Set

$$w_i(t) = \int_0^t s_i(x)dx = \sum_{k=1}^{j-1} \int_{r_{ik}}^{d_{ik}} s_i(x)dx + \int_{r_{ij}}^{t} s_i(x)dx = (j-1)\cdot C_i + c_i(t), \tag{6}$$

$$c_i(t) = \begin{cases} \min\{[t-r_{ij}]^+, C_1\} & \text{if } i=1, t\in[r_{ij},d_{ij}) \text{ and } m_i(t)\le C_i, \\ \min\{[Z_{i-1}(t)-Z_{i-1}(r_{ij})]^+, C_i\}, & \text{if } i>1, t\in[r_{ij},d_{ij}), \\ 0, & \text{otherwise;} \end{cases} \tag{7}$$

$$W_i(t) = W_{i-1}(t) + w_i(t),$$

$$Z_i(t) = Z_{i-1}(t) - w_i(t),$$

$$\mathcal{S}_i^{\mathrm{L}}(t) = \sum_{k=1}^{i} k\cdot s_k(t).$$

FPF dispatches the jobs to the processor as *early* as possible. The schedule for τ_i (5) reflects this by assigning jobs τ_{ij}, where $j = n_i(t)$, to the processor within the first C_i units of idle time in the interval $[r_{ij}, d_{ij})$. The quantity $c_i(t)$, defined by Eq. (7), determines the total service that job τ_{ij} has already received within its latest incomplete period at time t, $[r_{ij}, t)$. In section III we develop algorithms that compute $W_i(t)$ and $c_i(t)$ exactly and very efficiently. Once a schedule (5) is decided upon, all other quantities for τ_i can be determined directly. Job start s_{ij} and f_{ij} finish times are given by,

$$s_{ij} = \min\{x : x \ge r_{ij},\ V_{i-1}(x) - W_{i-1}(x) = 0\}, \tag{8}$$

$$f_{ij} = \min\{x : x \ge r_{ij} + C_i,\ Z_{i-1}(x) - Z_{i-1}(r_{ij}) = C_i\}, \tag{9}$$

which can be evaluated either via a simple fixed-point iteration algorithm or via bisection, within each interval $[r_{ij}, d_{ij})$.

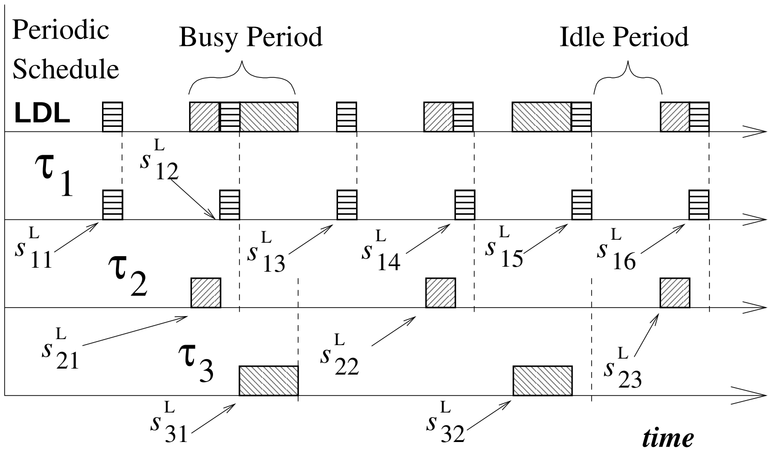


Fig. 3. The Idling Latest-Deadline-Last Schedule of the Task Set of Fig. 2

C.2 The Latest-Deadline-Last (LDL) Idling Schedule

We now consider a fixed priority, *idling* schedule for hard real-time periodic tasks we call *Latest-Deadline-Last* (LDL). Idling schedules have been known in the past but they could not be quantified efficiently, resulting algorithms of pseudo-polynomial complexity and high run-time overheads. In this schedule, denoted by $\mathcal{S}^{\mathrm{L}}$, we use the same fixed priority assignment as its FPF counterpart. Jobs τ_{ij} arrive at their release times r_{ij}, but they are *delayed*, each one by a different amount of time, after which they start again being dispatched by the regular preemptive, fixed-priority rules. LDL starts jobs as *late* as possible, while none misses any deadline. Fig. 3 shows the LDL schedule of the the same task set with Fig. 2. Notice that released jobs τ_{ij} are delayed until times s^{L}_{ij}. We will show in the sequel how we can utilize the newly created idle periods to accommodate firm aperiodic tasks. We formally derive $\mathcal{S}^{\mathrm{L}}$, and $s_i(t), W_i(t), Z_i(t),\ i = 1, 2, \ldots, n$. We proceed inductively, task-by-task, starting with the schedule $s_1(t),\ t > 0$, of τ_1 and then continuing with $\tau_2, \ldots, \tau_n$. Specifically, setting $V_0(t) = W_0(t) = 0$, for each $t \geq 0$, and $j = n_i(t)$, let

$$s_i(t) := \begin{cases} 1, & \text{if } i = 1, \text{ for } t \in [d_{ij} - C_1, d_{ij}), \\ 1, & \text{if } i > 1,\ \ Z_{i-1}(d_{ij}) - Z_{i-1}(t) < C_i, \text{ and } \mathcal{S}^{\mathrm{L}}_{i-1}(t) = 0, \\ & \text{for } t \in [r_{ij}, d_{ij}), \\ 0, & \text{otherwise;} \end{cases} \tag{10}$$

$$w_i(t) = \int_0^t s_i(s)ds = \sum_{k=1}^{j-1} \int_{r_{ik}}^{d_{ik}} s_i(x)dx + \int_{r_{ij}}^{t} s_i(x)dx = (j-1)\cdot C_i + c_i(t) \tag{11}$$

$$c_i(t) = \begin{cases} C_i - \min\{[d_{ij}-t]^+, C_i\}, & \text{if } i=1, t\in[r_{ij}, d_{ij}) \text{ and } (D_i - C_i) \le m_i(t) \le D_i, \\ C_i - \min\{[Z_{i-1}(d_{ij}) - Z_{i-1}(t)]^+, C_i\} & \text{if } i>1, t\in[r_{ij}, d_{ij}), \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

As usual, $W_i(t) = W_{i-1}(t) + w_i(t)$, $Z_i(t) = Z_{i-1}(t) - w_i(t)$ and $\mathcal{S}_i^{\mathrm{L}}(t) = \sum_{k=1}^{i} k \cdot s_k(t)$. As Eq. (10) indicates, LDL allocates the *last* C_i units of idle processor capacity in the interval $[r_{ij}, d_{ij})$, to job τ_{ij}. Effectively, this *delays* each job for as long as possible, without letting it miss its deadline. When τ_{ij} is released at time r_{ij} the dispatcher puts it into a *delay queue* until job start time s_{ij}. At that time the dispatcher starts considering it again for regular fixed-priority dispatching. In order to implement the LDL scheduler, we need to remember the job start times. We can obtain start s_{ij} and finish f_{ij} times, respectively, solving,

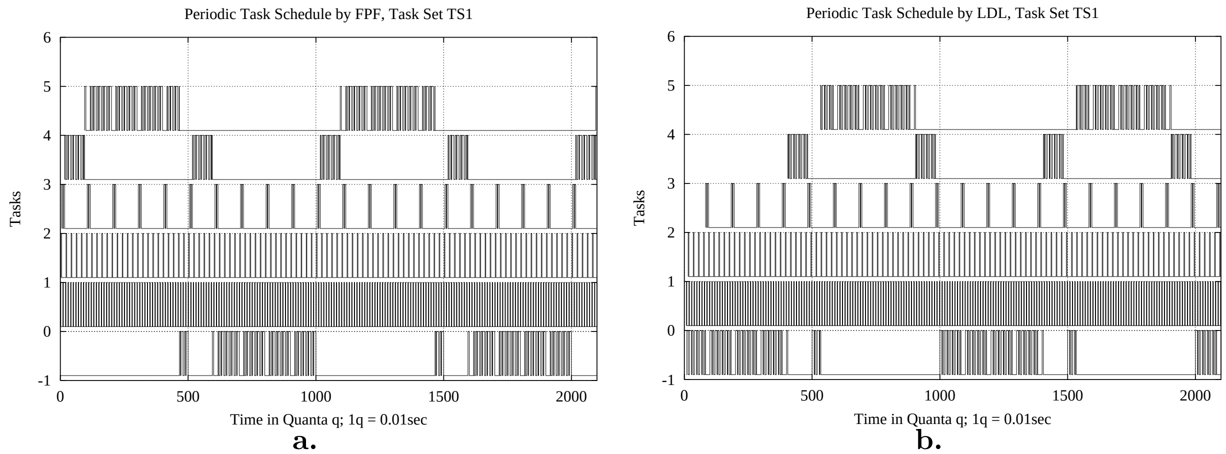
$$s_{ij} = \max\{x : x \le d_{ij},\ Z_{i-1}(d_{ij}) - Z_{i-1}(x) = C_i\}, \tag{13}$$

$$f_{ij} = \min\{x : x \ge (s_{ij} + C_i),\ Z_{i-1}(x) - Z_{i-1}(s_{ij}) = C_i\}, \tag{14}$$

for x with a fixed-point iteration or bi-section. If $\boldsymbol{\tau}$ is schedulable, the iteration converges, for discrete time schedules in at most $(d_{ij} - r_{ij})$ steps, and for continuous time schedules in finite time, as we do not allow instantaneous processor assignments.

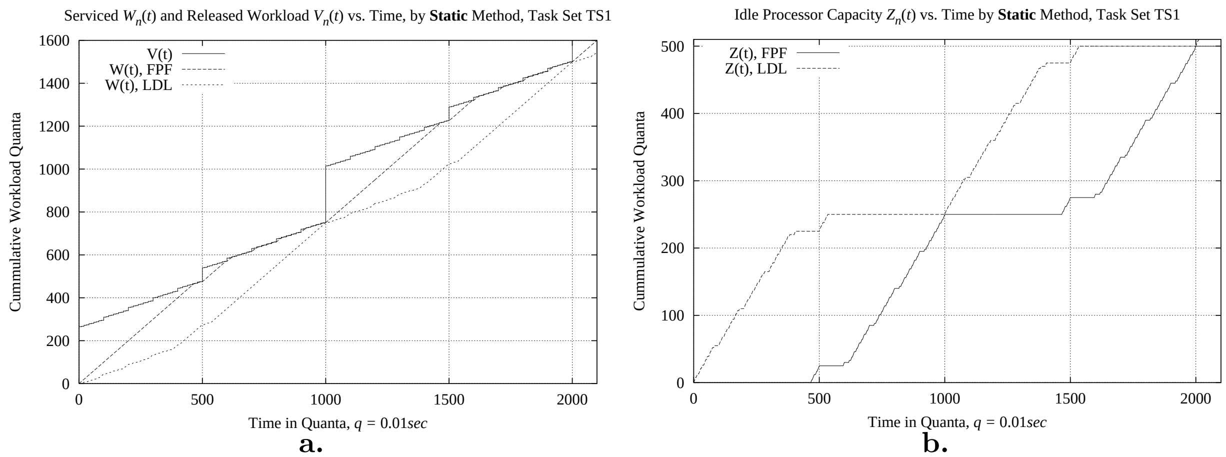
C.3 FPF and LDL Schedules of Task Set TS1

In this subsection we present the FPF and LDL schedules and the corresponding workload processes of task set TS1, which is defined in section D of the Appendix. Fig. 4**a.** presents the FPF and **b.** the LDL schedules, respectively. The bottom curves ("task"τ_0) show the *idle* periods. One readily notices that under FPF, most of the idle time concentrates towards the deadlines of the job instances, whereas under LDL it concentrates towards the job release times. An aperiodic task J_a arriving, for instance, at $r_a = 0$ and with deadline at $d_a = 400$, would have to be rejected outright due to the complete absence of idle capacity during $[0, 400)$. Fig. 5**a.** plots the released $V_n(t)$ and the serviced $W_n(t)$ workload under FPF (second from top curve), in the first two hyper-periods ($H = 1000$) of the same synthetic task set TS1.The release of new periodic jobs forces the upward jumps to the $V_n(t)$ process. $W_n(t)$ under FPF becomes increasing with a slope of a 45^o angle, as soon as new periodic tasks are released. $W_n(t)$ remains increasing while the unfinished workload $V_n(t) - W_n(t) > 0$. Fig. 5**b.** plots the corresponding idle capacity process $Z_n(t)$ (bottom curve). As



a. **b.**

Fig. 4. The FPF (**a.**) and LDL (**b.**) Schedules for Periodic Task Set TS1, in the First 2 Hyper-periods ($H = 10sec$)



a. **b.**

Fig. 5. **a.** The Serviced Workload $W_n(t)$ under FPF (Middle) and under LDL (Bottom) and the Released Workload $V_n(t)$ (Top) *vs.* Time. **b.** The Idle Capacity Process $Z_n(t)$ *vs.* Time, Under LDL (Top) and FPF (bottom). Synthetic Periodic Task Set TS1, the First two Hyper-periods ($H = 10sec$)

expected, $Z_n(t)$ remains parallel to x-axis while the processor services the periodic requests without any delay. When the unfinished workload $V_n(t) - W_n(t)$ at $t = 461$ becomes zero, the processor becomes idle and $Z_n(t)$ starts accumulating idle capacity.

Fig. 5**a.** plots the serviced $W_n(t)$ workload under LDL (bottom curve), in the first two hyper-periods ($H = 1000$) of the same synthetic periodic set TS1. Under FPF, $W_n(t)$ (middle curve in Fig. 5**a.**) starts increasing immediately after the release of periodic jobs. It remains increasing with slope 1 while the unfinished workload $V_n(t) - W_n(t) > 0$. Under LDL the serviced workload Initially progresses at a slower pace but as it approaches the end of a hyper-period it becomes increasing with slope equal to 1. Fig. 5**b.** shows the corresponding idle capacity curves $Z_n(t)$ under LDL (top) and FPF (bottom). We notice that, contrary to its FPF counterpart, LDL idle capacity accumulates “faster” towards the job release times.

C.4 Hybrid Schedules

FPF and LDL are the two extreme schedules for a periodic task set. It is possible to define fixed-priority schedules in which some of the tasks of $\boldsymbol{\tau}$ are dispatched according to FPF and the rest according to LDL. We are interested in the particular case where the first k tasks, $\tau_1, \dots, \tau_k$, follow the FPF, while the remaining $n-k$ tasks, $\tau_{k+1}, \dots, \tau_n$, follow the LDL schedules, respectively. Formal definition of such a hybrid schedule is possible by letting the first k tasks be characterized by the $s_i(t), w_i(t)$ from Eq. (5), (6) and the remaining $n-k$ tasks by those of Eq. (10) and (11).

D. Schedule and Workload Switching

D.1 Workload Theorems

Theorem 1 (Workload Dominance) Given a periodic task system $\boldsymbol{\tau}$, let $W_i^{\mathrm{F}}(t)$ and $W_i^{\mathrm{L}}(t)$ denote the serviced workload functions under the $\mathcal{S}^{\mathrm{F}}$ (FPF) and $\mathcal{S}^{\mathrm{L}}$ (LDL) schedules, respectively. The following holds

$$W_i^{\mathrm{F}}(t) \geq W_i^{\mathrm{L}}(t), \tag{15}$$

for all $i = 1, 2, \dots, n$ and $t > 0$.

Proof: See Section C in Appendix. ■

Corollary 1: Let $Z_i^{\mathrm{F}}(t)$ and $Z_i^{\mathrm{L}}(t)$ denote the idle capacity functions on the $\mathcal{S}^{\mathrm{F}}$ and $\mathcal{S}^{\mathrm{L}}$ schedules, respectively. Then, for all $t \geq 0$ and all $i = 1, 2, \dots, n$, the following is true,

$$Z_i^{\mathrm{F}}(t) \leq Z_i^{\mathrm{L}}(t). \tag{16}$$

Proof: See Section C in Appendix. ■

Theorem 2 (Workload Switch) Given a periodic task system $\boldsymbol{\tau}$, assume that the system dispatches the periodic tasks in the interval $[0, a)$ under FPF, at a it starts dispatching the tasks by the LDL schedule and it continues to do so until time $t > a$. Then the serviced workload in $[a, t)$, denoted as $dW_n^{\mathrm{FL}}(a, t)$, is less than or equal to the workload serviced had the system remained under the FPF schedule, denoted by $dW_n^{\mathrm{FF}}(a, t)$, that is

$$dW_i^{\mathrm{FL}}(a, t) \leq dW_i^{\mathrm{FF}}(a, t) \tag{17}$$

for all $i = 1, 2, \dots, n$ and $t \geq a \geq 0$.

Proof: See Section C in Appendix. ■

Corollary 1 and Theorem 2 affect the performance of any admission control scheme of a hard real-time system. In the absence of aperiodic tasks, periodic jobs are dispatched according to the FPF method. When a firm aperiodic $J_a(C_a, D_a, r_a)$ arrives, if $Z_n^{\mathrm{F}}(d_a) - Z_n^{\mathrm{F}}(r_a) < C_a$, where $d_a = r_a + D_a$, there is no sufficient idle capacity to guarantee J_a. However, by Theorem 2, $d_a - r_a - dW_n^{\mathrm{FL}}(r_a, d_a) \geq d_a - r_a - dW_n^{\mathrm{FF}}(r_a, d_a)$ making possible that $d_a - r_a - dW_n^{\mathrm{FL}}(r_a, d_a) \geq C_a$, thus, making J_a admissible. Intuitively, Theorem 2 suggests that, if until time r_a periodic tasks execute under FPF but within the time interval $[r_a, d_a)$ execute under LDL, the quantity of idle time within $[r_a, d_a)$ can only *increase* or (stay the same). LDL delays the execution of periodic tasks with the objective to "relocate" the idle capacity to temporal locations that may be needed by firm aperiodic tasks. Due to space limitations we have emphasized the on-line feasibility part of the the scheduling of one firm task only. The same feasibility testing algorithm has been succesfully incorporated into the on-line scheduling of firm tasks.

We note that the idea of delaying periodic tasks in order to accomodate aperiodic tasks is intuitively clear and not new. Several algorithms, such as slack-stealing and EDL, have attempted to utilize delayed schedules. However, these algorithms micro-manage the idle capacity continuously, giving algorithms with high complexity and overheads. We need to note that when the schedule returns back to FPF, the guaranteed LDL time to tasks should not be affected. Otherwise, the re-activation of delayed jobs, may render incomplete LDL jobs, as non-schedulable. Due to space limitations we cannot provide all of the details here.

D.2 Utilization of Gain Time

Periodic tasks τ_i are guaranteed off-line based on their maximum computation requirement C_i. Often in practice a job does not require to execute for its entire worst case time and terminates earlier. Time allocated but not used at run-time is called *gain time*. If gain time is not reused by other tasks, severe system under-utilization can result in. In the case of firm aperiodic tasks, leaving gain time un-utilized may result in several perfectly feasible firm tasks getting rejected. It is relatively straigforward to extend our method to handle gain time in the computation of serviced workload for currently active periodic tasks. The system maintains for each released job τ_{ij} the total service it has already received in variables $\xi_1, \ldots, \xi_n$. As soon as the system knows that an active job τ_{ij} requires g_{ij} time less than C_i it updates ξ_i by setting it to $\xi_i = \xi_i + g_{ij}$. Due to space limitations we cannot present here all of the details in the way gain times are utilized. A full version technical-report will be made available at the site of our department.

D.3 Increase of Idle Capacity with Schedule Switching

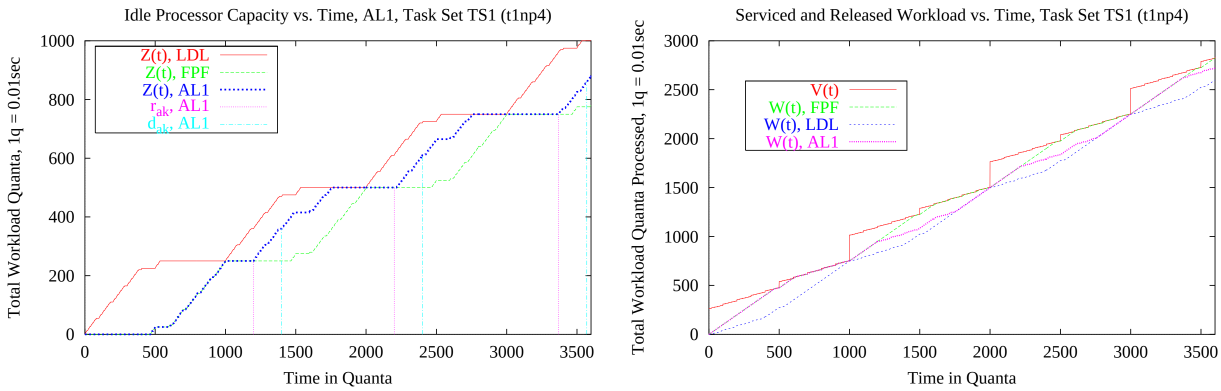


Fig. 6. The Idle Processor Capacity $Z(t)$ and Serviced Workload $W(t)$ Processes for Task Set TS1 during Schedule Switching

Here we illustrate the increase of idle processor capacity as the jobs of periodic set TS1, is switched twice from FPF to LDL, and then back to FPF. Fig. 6 presents $Z_n(t)$ and the corresponding $W_n(t)$ processes, during the schedule switches. In Fig. 6 the top and the bottom curves represent the idle process Z_n^{L} of LDL and Z_n^{F} of FPF, respectively. Before any non-periodic task arrives at the system, periodic tasks run under the FPF schedule. At time instant $t = 1200$, when the schedule is switched from FPF to LDL for the first time, the available processor capacity begins to increase, as the middle line shows, between the top and bottom curves in Fig. 6. When at $t = 1200$ periodic tasks switch to the LDL rule, no task misses its deadline. After the task set is switched back (from LDL) to FPF at time $t = 1400$, the idle capacity function resumes its growth pattern of the FPF mode. Similar results are observed when the non-periodic task is inserted for the second time, during the time interval $[2200, 2400]$. Fig. 6 shows the corresponding changes on the workload process. Until time $t = 1200$, the system services the periodic tasks as early as possible under the FPF schedule (top curve). It is worth noting that, in the initial FPF schedule (Fig. 5) there is no available capacity in $[1000, 1466]$ or in $[2000, 2466]$, at all. Any aperiodic task with arrival and deadline time within these intervals would had to be rejected. We notice in Fig. 6 that when we switch at $t = 1200$ and $t = 3200$ to LDL, idle capacity increases in these intervals. Switching from non-idling to idling schedules has been attempted in a number of algorithms in the past. However, all these algorithms require pseudo-polynomial time to make and implement on-line scheduling decisions. Having derived formally the extreme schedules FPF and LDL, the switching from one into the other, and the workload processes quantifying them, we are ready to derive linear time on-line feasibility algorithms, based upon the efficient computation of the workload processes.

III. Workload Matrix-based $W_i(t)$ Computation Algorithms

This section presents the *Workload Matrix* (WMATRIX) method for the direct computation of $W_i(t)$. In [12] and [13] two other methods were derived, one that applies to schedules with discrete task parameters and the second one which is based on a recursive computation. The WMATRIX algorithm does not require the task parameters to be integers, unlike other methods proposed in the past. Those methods proceed by first discretizing the time line, resulting an explosion in the size of hyper-period H. WMATRIX does not suffer from this drawback. It is well known that periodic task schedules repeat identically in each hypeperiod [14], [1]. Our algorithms need to analyze, for synchronous task sets the schedule of the first hyper-period in $[0, H]$, and for asynchronous sets, the schedule within $[0, r + 2H]$, where $r = \max_{i=1}^{n}\{r_i\}$.

From Eq. (6), (7) for FPF and Eq. (11), (12) for LDL of Section II-C, the serviced workload process in given by,

$$W_i^S(t) = \sum_{k=1}^{i}(n_k(t) - 1) \cdot C_k + c_k^S(t), \tag{18}$$

where $S = \mathrm{F}$ (FPF), or $S = \mathrm{L}$ (LDL). The most critical part in the computation is $c_k^S(t)$, defined as the amount of completed work for job $\tau_{k,n_k(t)}$ in the current request period at time t. Specifically, for $k \geq 1, t \in [r_{k,n_k(t)}, d_{k,n_k(t)})$,

$$\begin{aligned} c_k^{\mathrm{F}} &= \min\big\{[Z_{k-1}^{\mathrm{F}}(t) - Z_{k-1}^{\mathrm{F}}(r_{k,n_k(t)})]^+, C_k\big\} \\ &= \min\big\{\big[(t - r_{k,n_k(t)}) - \big(W_{k-1}^{\mathrm{F}}(t) - W_{k-1}^{\mathrm{F}}(r_{k,n_k(t)})\big)\big]^+, C_k\big\} \end{aligned} \tag{19}$$

and

$$\begin{aligned} c_k^{\mathrm{L}} &= C_k - \min\big\{[Z_{k-1}^{\mathrm{L}}(d_{k,n_k(t)}) - Z_{k-1}^{\mathrm{L}}(t)]^+, C_k\big\} \\ &= C_k - \min\big\{\big[(d_{k,n_k(t)} - t) - \big(W_{k-1}^{\mathrm{L}}(d_{k,n_k(t)}) - W_{k-1}^{\mathrm{L}}(t)\big)\big]^+, C_k\big\} \end{aligned} \tag{20}$$

where $W_0^{\mathrm{L}}(t) := 0$ for all t.

Direct fixed-point computation based on Eq. (19) and (20) leads to exponential time computation due the double-recurrence on $c_k^S(t)$. Upon closer examination it becomes clear that $c_k^{\mathrm{F}}(t)$ depends on W_{k-1}^{F} evaluated at points t and $r_{k,n_k(t)}$, which are the release times of the jobs $\tau_{k,n_k(t)}$ and are well defined. We ready now to derive an efficient on-line $W_n^{\mathrm{F}}(t)$ computation procedure, having two steps. First, we pre-compute off-line $W_{k-1}^{\mathrm{F}}(r_{k,j_k})$, for $k = 2, 3, \ldots, n,\ j_k = 1, 2, \ldots, n_k(H)$, and store them in a matrix $\boldsymbol{W}^{\mathrm{F}}[k, j_k]$. We need not store W_1^{F} which can be computed directly. At run-time, setting $W_0^{\mathrm{F}}(t) = \boldsymbol{W}^{\mathrm{F}}[0, j_0] = 0$, we compute $W_n^{\mathrm{F}}(t)$ by

$$W_n^{\mathrm{F}}(t) = \sum_{k=1}^{i} (n_k(t) - 1) \cdot C_k + c_k^{\mathrm{F}}(t)$$
$$c_k^{\mathrm{F}}(t) = \min\Big\{ \big[(t - r_{k,n_k(t)}) - (W_{k-1}^{\mathrm{F}}(t) - \boldsymbol{W}^{\mathrm{F}}[k, n_k(t)])\big]^+, C_k \Big\}. \tag{21}$$

The time complexity of the WMATRIX method is $T_W^{\mathrm{wm}}(n) = \Theta(n)$. The storage complexity $S_W^{\mathrm{wm}}(n) = \Theta(\sum_{k=2}^{n} n_k(H)) = N$. We can even avoid storing the $W_2^{\mathrm{F}}(r_{3,j_k})$ since W_2^{F} can be directly computed.

In a similar fashion under LDL, $c_k^{\mathrm{L}}(t)$ depends on W_{k-1}^{L} evaluated at points t and $d_{k,n_k(t)}$, which are the deadlines of the jobs $\tau_{k,n_k(t)}$ and are also well defined. The LDL WMATRIX algorithm similarly pre-computes off-line $W_{k-1}^{\mathrm{L}}(d_{k,j_k})$ for $k = 2, 3, \dots, n,\ j_k = 1, 2, \dots, n_k(H)$, and stores them in a matrix $\boldsymbol{W}^{\mathrm{L}}[k, j_k]$. At run time, setting $W_0^{\mathrm{L}}(t) = \boldsymbol{W}^{\mathrm{L}}[0, j_0] = 0$, it computes $W_n^{\mathrm{L}}(t)$ by

$$W_n^{\mathrm{L}}(t) = \sum_{k=1}^{i} (n_k(t) - 1) \cdot C_k + C_k - \bar{c}_k^{\mathrm{L}}(t)$$
$$\bar{c}_k^{\mathrm{L}}(t) = \min\Big\{ \big[(d_{k,n_k(t)} - t) - (\boldsymbol{W}^{\mathrm{L}}[k, n_k(t)] - W_{k-1}^{\mathrm{L}}(t))\big]^+, C_k \Big\}. \tag{22}$$

This method has time complexity $T_W^{\mathrm{wm}}(n) = \Theta(n)$, and storage complexity $S_W^{\mathrm{wm}}(n) = \Theta(\sum_{k=2}^{n} n_k(H))$, as its FPF counterpart. To our knowledge this is the first linear time algorithm that computes idle capacity at any arbitrary point t, exactly. The WMATRIX methods has very low actual execution time as the experiments in Section IV demonstrate, making it an excellent algorithm for the feasibility testing of firm aperiodic tasks.

IV. Performance Evaluation of WMATRIX

A. *Admission Probability Experiments*

In this set of experiments we measure the probability for admission $\mathrm{P}_{\mathrm{adm}}$ of firm aperiodic tasks J_{ak} under different aperiodic task workload conditions. For each aperiodic task J_{ak} the scheduler computes at $t = r_a$ the amount of idle capacity of the periodic schedule in the interval $[r_a, d_a]$, where $d_a = r_a + D_a$. If the periodic tasks are currently serviced under the FPF and there is sufficient capacity ($Z_n^{\mathrm{F}}(r_a, d_a) \geq C_a$), the scheduler admits J_a. If there is no sufficient capacity, the scheduler evaluates the idle capacity as if a schedule switch would take place at r_a, evaluating $Z^{\mathrm{FL}}(r_a, d_a) = D_a - dW^{\mathrm{FL}}(r_a, d_a)$. If $Z^{\mathrm{FL}}(r_a, d_a) \geq C_a$ it accepts J_a, otherwise, the scheduler rejects J_a. The results presented here are based on one firm task being out-standing at a time. We will present the guaranteed scheduling algorithms of multiple pending firm tasks, under various service disciplines, elsewere.

In Fig. 7 through 10 we plot a smoothed average of the admission probability $\mathrm{P}_{\mathrm{adm}}(\rho_a)$ under varying load conditions with respect to the arrival rate ρ_a of the firm tasks. Here, $\rho_a := \mathrm{E}\left[\frac{\bar{C}_a}{\bar{T}_a}\right]$, where $\bar{C}_a$ is the expected computation requirements of firm tasks, and $\bar{T}_a$ is the expected task inter-release times. Both computation and inter-arrival times follow the exponential distribution with parameters, $\bar{C}_a$ and $\bar{T}_a$, respectively. We evaluate the performance of the admission control procedure under a wide range of aperiodic workload, as follows. For each value of $\rho_a = 0.1, 0.2, \ldots, 0.9$ we vary $\bar{T}_a$ in the range $[1, H)$ and compute the corresponding $\bar{C}_a = \rho_a \cdot \bar{T}_a$. For particular values of $\bar{C}_a$ and $\bar{T}_a$, we generate a sequence of tasks J_{ak}, setting as deadline $D_{ak} = r_{a,k+1} - r_{ak}$. D_{ak} is the inter-arrival time of the next task $J_{a,k+1}$. From the simulation experiments we ignore values $C_{ak} > D_{ak}$.

We use different periodic task sets[2] to run feasibility experiments and measure $\mathrm{P}_{\mathrm{adm}}(\rho_a)$ against the different $\bar{C}_a$ and $\bar{T}_a$ values, as described above. Fig. 7 through 10 plot the $\mathrm{P}_{\mathrm{adm}}(\rho_a)$ against the different ρ_a values. Each figure shows two curves corresponding to the two different admission control procedures. The lower curve shows the admission probability when the system operates continuously under the FPF schedule. The upper curve shows the admission probability when the same sequence of aperiodic tasks are tested for feasibility and the system switches to the LDL schedule in case the FPF schedule could not guarantee a task J_{ak}. We clearly see that under all possible workload conditions we can guarantee more aperiodic tasks when the system switches into the LDL schedule. As LDL is the latest schedule that relocates the idle capacity in any interval $[r_{ak}, d_{ak})$ this is also the only schedule that maximizes the probability of admission $\mathrm{P}_{\mathrm{adm}}(\rho_a)$ for a sequence of aperiodic tasks J_{ak}. The robustness of the feasibility testing method stems from the fact that using the WMATRIX method we can quantify exactly both FPF and LDL schedules in a very computationally efficient way.

B. Measurements of Workload Matrix Generation Times and Sizes

Here we present the time it takes to generate off-line the workload-matrix on a $333MHz$ Ultra-Sparc 5 SUN workstation, for each one of the example task sets used in our experiments. Table I presents the generation times in seconds and the size of each matrix in memory locations. We notice that for modest task sets the matrix generation time is relatively low. This makes it possible to even generate a matrix on-line, in case dynamic adjustments in the periodic task set is necessary.

[2]The periodic task sets of the experiments are described in Section D of the Appendix.

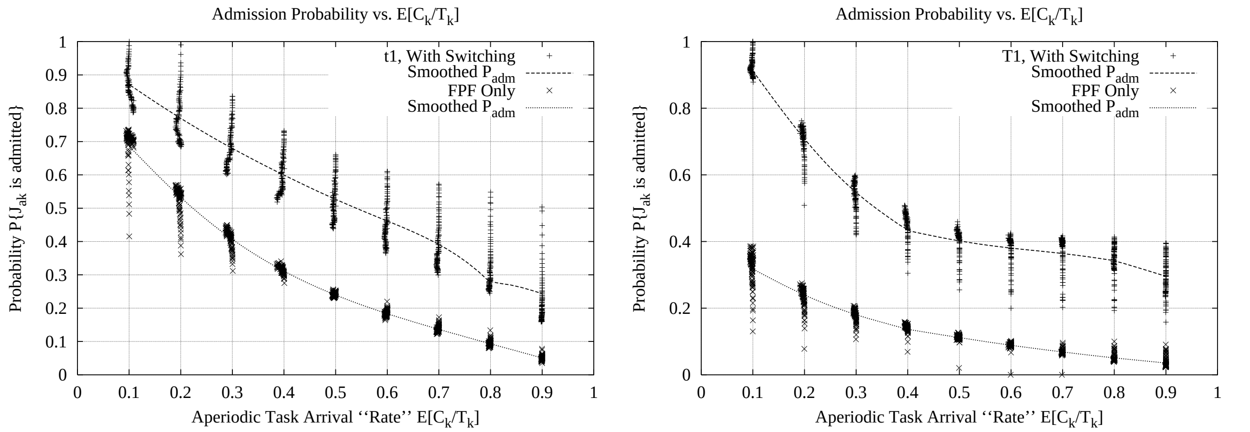


Fig. 7. Admission Probability P_{adm} with Selective Schedule Switching and under Guaranteed FPF, for Task Sets TS1 and T1

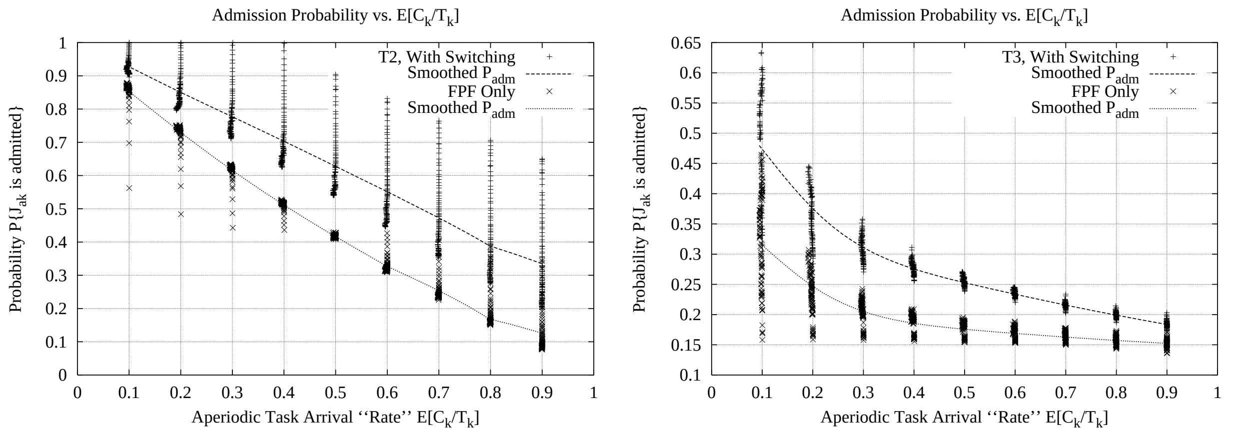


Fig. 8. Admission Probability P_{adm} with Selective Schedule Switching and under Guaranteed FPF, for Task Sets T2 and T3

C. Feasibility Testing Run-Time Overhead

In this set of experiments we perform $W_n^{\mathrm{F}}(t)$ and $W_n^{\mathrm{L}}(t)$ computations on several UNIX machines, and measure the actual execution times. The computation of $W_n^S(t)$ is the time the system needs to make the on-line feasibility testing decision for the incoming task J_{ak}. The times for $W_n^{\mathrm{F}}(t)$ and $W_n^{\mathrm{L}}(t)$ are identical. Here we report the ones for $W_n^{\mathrm{F}}(t)$ only. In Fig. 11 we plot the execution time as the number of periodic tasks increases, from $n = 1$ to $n = 16$ for the Periodic Sets of Collection 2 (those in the Appendix). With 16 tasks a 250 MHz SGI workstation, required 2.8μ-seconds, whereas, a 143MHz SUN Ultra-Sparc I required 8.5μ-seconds.

V. Conclusions

The WMATRIX is an efficient method for the quantification of the workload of periodic tasks in hard real-time, fixed priority systems. It is coupled with the switching from the standard, non-

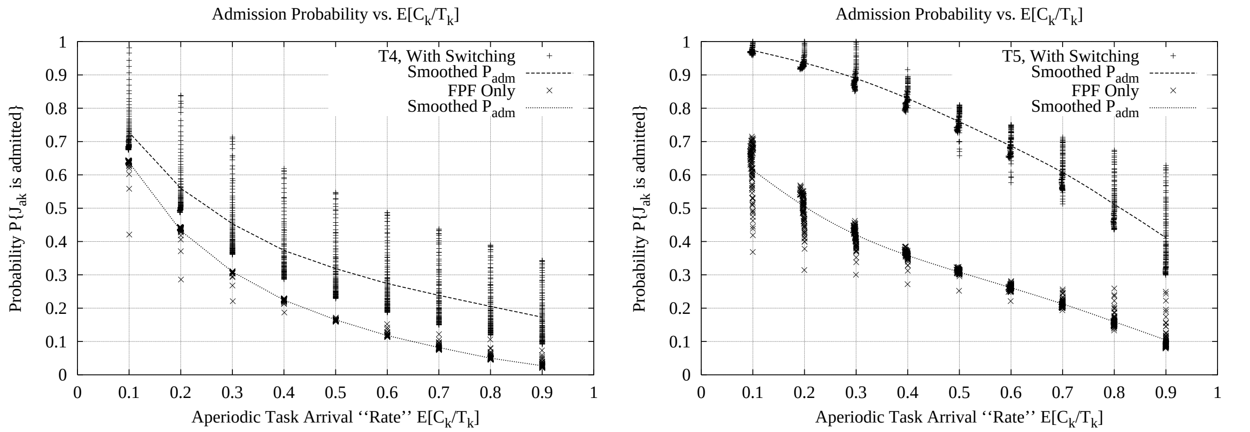


Fig. 9. Admission Probability P_{adm} with Selective Schedule Switching and under Guaranteed FPF, for Task Sets T4 and T5

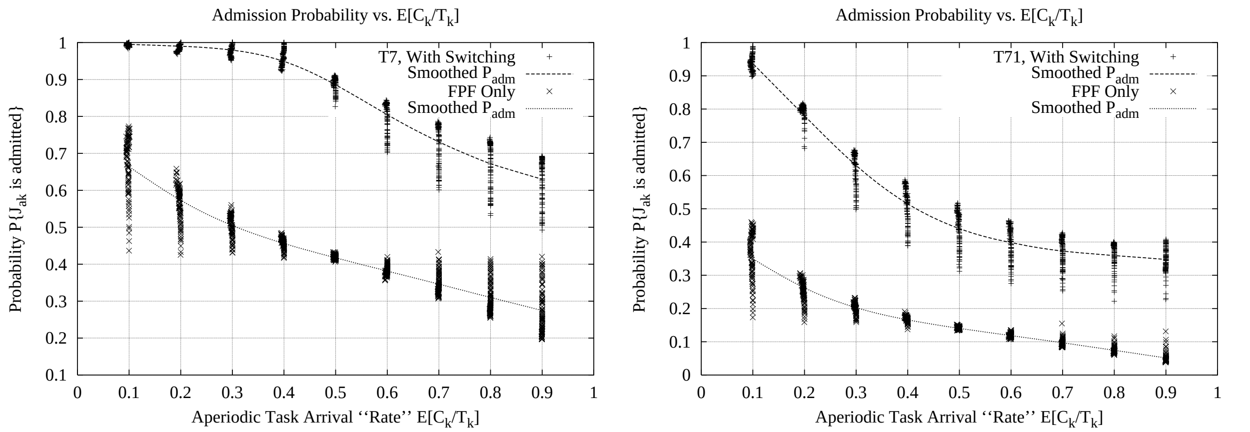


Fig. 10. Admission Probability P_{adm} with Selective Schedule Switching and under Guaranteed FPF, for Task Sets T7 and T71

idling FPF into the idling LDL periodic task schedules. It is the first linear time, on-line, exact feasibility testing algorithm reported in the literature that maximizes the probability of admission of firm aperiodic tasks. We have shown through analysis and experiments that admission control based on WMATRIX has very low asymptotic and execution time cost. We have presented the scheduling of one outstanding firm task at a time, in order to focus on the on-line feasibility testing aspect of the algorithm. Other methods with similar objectives, such as, Slack-Stealing and EDL methods require pseudo-polynomial time and incur very high run-time overheads. The WMATRIX method has been succesfully incorporated into scheduling algorithms for large, complex and highly dynamic real-time systems where new tasks need to be guaranteed on-line very efficiently. One drawback of WMATRIX is the matrix size. We are developing approximate feasibility algorithms which can use a smaller matrix at the expense of exactness. However, preliminary results show that in most cases the difference in admission probability is small. In environments were memory space is at a premium, keeping the matrix at a manageable size is more important than maximizing the

TABLE I

Workload-Matrix Generation Times and Matrix Sizes

Name	n	Time	Size	Name	n	Time	Size
TS1	5	0.000063289	164	TS2	5	0.000090236	204
TS3	10	0.001493509	2373	T1	6	0.000162340	351
T2	6	0.000147966	270	T3	11	0.000106712	111
T4	15	0.027065545	21656	T5	11	0.000669799	1897
T7	10	0.000156195	284	T71	13	0.000701562	748

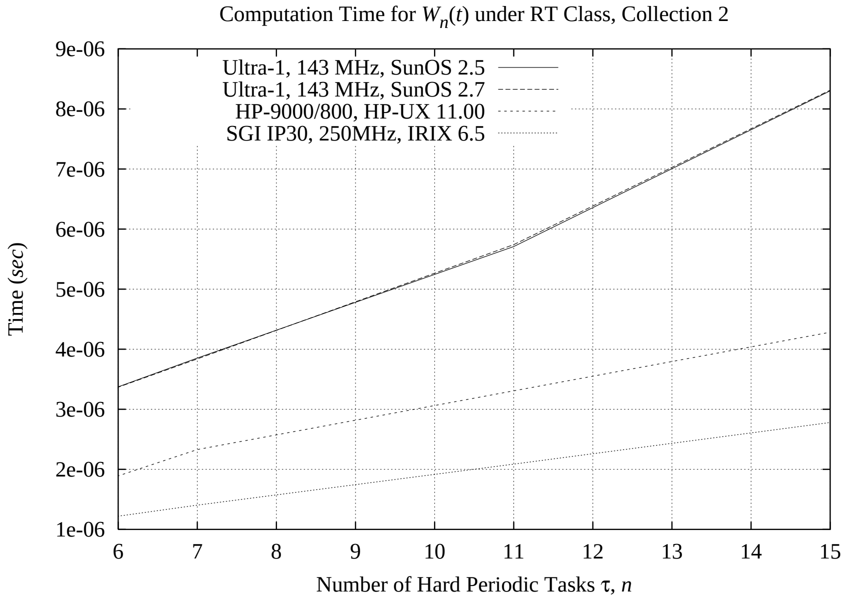


Fig. 11. Computation Time of $W_n(t)$ for each of the Periodic Sets of Collection 2 under the Real-Time Scheduling Class in UNIX Platforms

admission probability. We note in passing that WMATRIX can be easiliy extended to distributed systems that load-sharing is used to mitigate the task rejection problem. In subsequent work we will present the details of the afore-mentioned extensions.

Finally, it is worth noting that schedulers based on dynamic priority assignment rules, including the Earliest-Deadline-First (EDF) and Least-Slack-First (LST), cannot have linear exact on-line admission algorithms. They need to check the schedulability of each new task within arbitrary periods of time, against a possibly pseudo-polynomial number of ready tasks. Our work has demonstrated that under fixed priority rules it is possible to guarantee tasks in linear time with exact schedulability testing algorithms.

References

[1] Joseph Y.-T. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic real-time tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[2] R. I. Davis, K. W. Tindell, and A. Burns, "Scheduling slack-time in fixed priority preemptive systems," in *Proc. 14th IEEE Real Time Systems Symp.*, Raleigh-Durham, NC, Dec. 1993, pp. 222–231.

[3] John P. Lehoczky and Sandra Ramos-Thuel, "Scheduling periodic and aperiodic tasks using the slack stealing algorithm," in *Advances in Real-Time Systems*, Sang H. Son, Ed., chapter 8, pp. 175–198. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1995.

[4] Houssine Chetto and Maryline Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Trans. on Software Engineering*, vol. 15, no. 10, pp. 1261–1269, Oct. 1989.

[5] Maryline Silly, "The EDL server for scheduling periodic and soft aperiodic tasks with resource constraints," *Real-Time Systems*, vol. 17, no. 1, pp. 87–111, July 1999.

[6] R. I. Davis, "Approximate slack stealing algorithms for fixed priority pre-emptive systems," Tech. Rep. YCS–93–216, Comp. Science Dep't, Univ. of York, UK, 1993.

[7] R. I. Davis, K. W. Tindell, and A. Burns, "Mechanisms for enhancing the flexibility and utility of hard real-time systems," in *Proc. 15th IEEE Real Time Systems Symp.*, San Juan, Puerto Rico, Dec. 1994, pp. 12–21.

[8] Charlie McElhone, A. Burns, and R. Davis, "Hybrid algorithms for dynamic schedulability testing," in *Proc. 7th Euromicro Workshop on Real-Time Systems*, 14–16 June 1995, pp. 254–261.

[9] C. M. Krishna and Kang G. Shin, *Real-Time Systems*, Mc Graw-Hill, New-York, NY, 1997.

[10] C. Liu and J. Layland, "Scheduling algorithms for multi-programming in a hard real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, 1973.

[11] Jane W. S. Liu, *Real-Time Systems*, Prentice-Hall, Upper Saddle River, New Jersey 07458, 2000.

[12] Michael E. Thomadakis, *Algorithms and Techniques for Joint Task Scheduling in Open Computing Systems*, Ph.D. thesis, Texas A&M University, College Station, TX 77843-3112, Dec. 1999.

[13] Michael E. Thomadakis and J-C. S. Liu, "Linear time on-line feasibility testing algorithms for fixed-priority, hard real-time systems," Tech. Rep. TR-2000-006, Dept. of Computer Science, Texas A&M University, College Station, TX 77843-3112, Jan. 21 2000.

[14] Joseph Y.-T. Leung and M. L. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, no. 3, pp. 115–118, 1980.

[15] J. Lehoczky, Lui Sha, J. K. Strosnider, and Hide Tokuda, "Fixed priority scheduling theory for hard real-time systems," in *Foundations of Real-Time Computing: Scheduling and Resource Management*, André M. van Tilborg and Gary M. Koob, Eds., chapter 1, pp. 1–30. Kluwer Academic Publishers, Norwell, MA, 1991.

[16] Lui Sha, Mark H. Klein, and John B. Goodenough, "Rate monotonic analysis for real-time systems," in *Foundations of Real-Time Computing: Scheduling and Resource Management*, André M. van Tilborg and Gary M. Koob, Eds., chapter 5, pp. 129–155. Kluwer Academic Publishers, Norwell, MA, 1991.

[17] J. Lehoczky, L. Sha, and J. K. Strosnider, "Enhanced aperiodic responsiveness in hard real-time environments," in *Proc. 8th IEEE Real Time Systems Symp.*, Dec. 1987, pp. 261–270.

[18] B. Sprunt, J. Lehoczky, and L. Sha, "Exploiting unused periodic time for aperiodic service using the extended priority-exchange algorithm," in *Proc. 9th IEEE Real Time Systems Symp.*, Dec. 1988, pp. 251–258.

[19] J. K. Strosnider, J. P. Lehoczky, and L. Sha, "The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments," *IEEE Trans. on Computers*, vol. 44, no. 1, pp. 73–91, Jan. 1995.

[20] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard real-time systems," *The Journal of Real-Time Systems*, vol. 1, pp. 27–60, 1989.

[21] Sandra Ramos-Thuel and John P. Lehoczky, "Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing," in *Proc. 15th IEEE Real Time Systems Symp.*, San Juan, Puerto Rico, Dec. 1994, pp. 22–33.

[22] R. I. Davis, "Scheduling slack time in fixed priority pre-emptive systems," Tech. Rep. YCS–93–217, Comp. Science Dep't, Univ. of York, UK, 1993.

[23] M. Joseph and P. Pandya, "Finding the response times in a real-time system," *Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[24] N. Audsley, A. Burns, R. I. Davis, and A. J. Wellings, "Integrating best effort and fixed-priority scheduling," in *Proc. of 1994 Workshop on Real-Time Programming*, June 1994, pp. 45–55.

[25] N. Audsley, R. I. Davis, A. Burns, and A. J. Wellings, "Appropriate mechanisms for the support of optional processing in hard real-time systems," in *Proc. 11th IEEE Real-Time Operating Systems and Software*, Seattle, WA, May 18–19 1994, pp. 23–27.

[26] Pamm Binns, "Incremental rate-monotonic scheduling for improved control system performance," in *Proc. Third IEEE Real-time Technology and Applications Symposium*, Montreal, Canada, June 9–11 1997, pp. 80–90.

[27] Charlie McElhone, "Adapting and evaluating algorithms for dynammic schedulability testing," Tech. Rep. YCS–94–225, Department of Computer Science, University of York, UK, 1994.

[28] Karsten Schwan and Hongyi Zhou, "Dynamic scheduling of hard real-time tasks and real-time threads," *IEEE Trans. on Software Engineering*, vol. 18, no. 8, pp. 736–748, Aug. 1992.

[29] Maryline Silly, Houssine Chetto, and Nadia Elyounsi, "An optimal algorithm for guaranteeing sporadic tasks in hard real-time systems," in *Second IEEE Symposium on Parallel and Distributed Processing*, 9–13 Dec. 1990, pp. 578–585.

[30] Housine Chetto, Maryline Silly, and T. Bouchentouf, "Dynamic scheduling of real-time tasks under precedence constraints," *The Journal of Real-Time Systems*, vol. 2, pp. 181–194, 1990.

[31] Too-Seng Tia, *Utilizing Slack Time for Aperiodic and Sporadic Requests Scheduling in Real-Time Systems*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, Apr. 1995, Available as Technical Report UIUCDCS-R-95-1906.

[32] Ismail Ripoll, Ana García-Formes, and Alfons Crespo, "Optimal aperiodic scheduling for dynamic-priority preemptive systems," *Proc. Third Int'l Workshop in Real-Time Computing Systems and Applications*, pp. 294–300, 1996.

[33] Ismail Ripoll, Alfons Crespo, and Ana García-Formes, "An optimal algorithm for scheduling soft aperiodic tasks in dynamic-priority preemptive systems," *IEEE Trans. on Software Engineering*, vol. 23, no. 6, pp. 388–400, June 1997.

[34] Giorgio C. Buttazzo and Fabrizio Sensini, "Optimal deadline assignment for scheduling soft aperiodic tasks in hard real-time environments," in *Proc. of the Third IEEE Int'l Conf. on Engineering of Complex Computing Systems*, Como, Italy, Dec. 1997, pp. 39–48.

[35] Giorgio C. Buttazzo and Marco Caccamo, "Minimizing aperiodic response times in a firm real-time environment," *IEEE Trans. on Software Engineering*, vol. 25, no. 1, pp. 22–32, Jan./Feb. 1999.

[36] Gerhard Fohler, "Joint scheduling of distributed complex periodic and hard aperiodic tasks in statically scheduled systems," in *Proc. 16-th IEEE Real-Time Systems Symp.*, Dec. 1995, pp. 152–161.

[37] Damir Isović and Gerhard Fohler, "Handling sporadic tasks in off-line scheduled distributed real-time systems," in *Proc. of the 11th Euromicro Conf. on Real-Time Systems*, 9–11June1999 1999, pp. 60–67.

[38] Damir Isović and Gerhard Fohler, "Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints," in *Proc. 21st IEEE Real-Time Systems Symp.*, 27–30Nov. 2000, pp. 207–216.

[39] Giorgio C. Buttazzo and J. A. Stankovic, "RED: Robust earliest deadline scheduling," Tech. Rep. UM–CS–1993–025, Comp. Science Dep't, Univ. of Massachusetts, Amherst, July 1993.

[40] Giorgio C. Buttazzo and J. A. Stankovic, "Adding robustness in dynamic preemptive scheduling," in *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*, D. S. Fussell and M. Malek, Eds. Kluwer Academic Publishers, Boston, MA, 1995.

Appendix

Note to the reviewers: we have included this Appendix for ease of reference. We do not plan to include this Appendix in the final manuscript in case it is accepted, but we will provide instead, a Tech-Report accessible at the site of our department with the full version of the paper.

I. Notation

TABLE II

Definition of Quantities

Notation	Definition
$\boldsymbol{\tau}$	Task system with n tasks $\tau_i(T_i, C_i, D_i, r_i)$
τ_{ij}	Job τ_{ij}, the jth instance of task τ_i
$\boldsymbol{J}_a$	Aperiodic job sequences
$v_i(t)$	Total workload for task τ_i in $[0, t]$
$V_i(t)$	Aggregate released workload for tasks $\tau_k,\ k = 1, 2, \dots, i$ in $[0, t]$
$\mathcal{S}(t)$	Schedule Mapping $\mathcal{S} : \mathbb{R} \longrightarrow \{0, 1, \dots, \mathrm{n}\}$
$s_i(t)$	Indicator function for task τ_i in schedule $\mathcal{S}(t)$
$w_i(t)$	Total serviced workload for task τ_i in $[0, t)$
$W_i(t)$	Aggregate serviced workload for tasks $\tau_k,\ k = 1, 2, \dots, i$ in $[0, t)$
$Z_i(t)$	Idle processor capacity in $[0, t)$, after tasks $\tau_k,\ k = 1, 2, \dots, i$ have been serviced
s_{ij}, f_{ij}	Start and finish time for τ_{ij}, with $r_{ij} \leq s_{ij}$ and $f_{ij} \leq d_{ij}$
R_{ij}	Response time of τ_{ij}, with $R_{ij} := f_{ij} - r_{ij}$
$n_i(t)$	$\left\lfloor \frac{t - r_i}{T_i} \right\rfloor + 1$, current period number of τ_i at time t
$m_i(t)$	$(t - r_i) \pmod{T_i}$, "translated" remainder
$[x]^+$	$\max\{x, 0\}$.

II. Previous Work on Feasibility Algorithms

A number of on-line feasibility testing algorithms have been proposed in the past. In this section we provide an overview of these methods and discuss their time and space complexities. We include feasibility testing methods for dynamic priority systems for comparison purposes. Feasibility algorithms fall broadly into two categories, namely, the *slack-stealing* and the *schedule construction* methods. A third approach, the *server-based* methods, preallocate off-line portions of the schedule via the use of aperiodic servers, such as, the *Poll* server (PS) [15], [16], the *Priority Exchange* (PE), the *Extended-Priority Exchange* (EPE) [17], [18], the *Deferrable* (DS) [17], [19], and the *Sporadic*

(SpS) [20] server. These methods represent important milestones as algorithms for the scheduling of aperiodic tasks. However, they are not feasibility algorithms *per-se*, and, thus, we do not discuss them any further.

Slack Stealing (SS) algorithms are based on the observation that a periodic task instance τ_{ij} can be delayed at time t by an amount of time equal to its *slack* $S_i(t)$ without missing its deadline. When slack is available at all priority levels $S_{\min} = \min_{i=1}^{n}\{S_i(t)\}$ then a firm aperiodic task J_a can take precedence over all tasks during $[t, t + S_{\min})$. To provide on-line guarantees to J_a the SS algorithms search all consecutive slack intervals until they can find total spare capacity exceeding the execution requirements C_a. This approach leads to pseudo-polynomial time algorithms. Even after J_a is admitted, SS continuously searches for the immediately next slack interval to grant to J_a until it finishes. Furthermore, slack correctness requires continuous maintenance of $\Theta(n+m)$ slack variables, at the completion of every task or at each task switching. The high admission overhead and the continuous state maintenance make SS algorithms very hard to scale as the number of aperiodic requests increases. Therefore, as the aperiodic tasks become more fine-grained, the system spends proportionally more time deciding if a firm is schedulable, than serving aperiodic requests.

Static Slack Stealing (SSS) [21], [3], upon arrival of a task J_a determines the amount of slack within time interval $[r_a, r_a + D_a]$ based on precomputed information about slack. Task admission and subsequent scheduling of J_a each take $O((H + D_a)n^2)$ time [21], where n is the number of periodic tasks and H is their hyper-period, which is pseudo-polynomial time. SSS algorithms place high run-time execution and storage requirements and require continuous maintenance of $\Theta(n+m)$ state variables. Davis *et al.*, [2], [22] proposed an exact *dynamic* slack stealing (EDSS) method. EDSS locates slack dynamically at run time, using a forward search, fixed-point iteration algorithm, based on a method proposed by Joseph and Pandya in [23] to calculate busy period lengths. EDSS combines the preprocessing step of the SSS with dynamic slack determination and, thus, has very high run-time computation overhead. It determines the amount of slack at all n priority levels, in time $O(G \cdot n^2)$. G is a linear function of the total number of busy periods in $[t, t + \max_i\{d_i(t)\})$, which is pseudo-polynomial. EDSS theoretically must run at each scheduler clock-tick to keep current the slack variables. It runs at least at the completion of each periodic task instance and it requires $O(n)$ slack variable maintenance at each context-switching time. Davis *et al.*, [6], [24], [25] also proposed approximate DSS (ADSS) algorithms which compute pessimistic estimates of the slack $S_i(t)$. Slack computation at all n levels either for admission control, and or for the times at which the firm aperiodic task can run, each take $O(n^2 + n)$ time. On-line admission control for one firm aperiodic with a heuristically preselected priority takes $\Theta(n + m)$ time. Again, the slack for particular task must be recomputed at the completion of each one of its instances in

$O(n)$ time, and $O(n+m)$ slack variables must be maintained at every context switching time. For larger periodic task sets the continuous slack maintenance becomes a significant source of overhead. Another problem is that in the proposed method the priority of a candidate aperiodic task J_a is heuristically preselected to be compatible with the priorities of the tasks that precede and follow J_a in deadline order. If the deadline D_a of J_a is larger than the deadlines D_i of all the periodic tasks, $D_a > \max_{i=1}^{n}\{D_i\}$, the EDSS and ADSS algorithms degrade into background processing. More recently, Binns [26] proposed algorithms based on SSS for the scheduling of sporadic tasks, resulting from *incremental* and *design-to-time* periodic tasks. An admission control decision for a sporadic task J_s at a priority level k, requires time $O(n) + (t_k - 1) \cdot \max\{O(n(n-1)), O(n \left\lceil \frac{T_n}{T_1} \right\rceil)\}$, where t_k is an integer number, the "transform" factor of T_k [26].

McElhone [27], [8] investigated two on-line algorithms which test the schedulability of one firm aperiodic task. The first one is pessimistic and takes $\Theta(n^2)$ time to compute a lower bound on the spare capacity. The second is an exact algorithm taking pseudo-polynomial time. When the deadline D_a of the firm task is $D_a > \max_{i=1}^{n}\{D_i\}$ of the deadlines of the periodic tasks, the algorithms essentially schedule the firm at background (lowest) priority.

For comparison purposes we also summarize the complexity of firm task scheduling based on dynamic priority algorithms. Chetto and Chetto [4] and Schwan and Zhou [28] investigated on-line admission tests for systems scheduled by the Earliest Deadline First (EDF) algorithm. Their methods require time proportional to the number N of all periodic jobs released within the least common multiple of the periodic task periods, called the hyper-period H, and thus, they are pseudo-polynomial in time complexity. Further, Schwan and Zhou methods require all periodic and aperiodic tasks to go through continuous schedulability testing, leading to unnecessarily high overheads. The method by Schwan *et al.*, is of the schedule construction type.

Silly and Chetto [29] proposed feasibility algorithms, based on the *EDL method*, to guarantee firm tasks, taking $O(N+m(r_a))$ time and storage $O(N)$, where N is the total number of invocations of all the periodic tasks within one hyper-period, and $m(r_a)$ is the number of pending aperiodic tasks at the time of admission. Chetto *et al.*, [30] presented an algorithm to admit groups of m firm aperiodic tasks related by precedence constraints. This algorithm takes $O(N^2)$ time where N is the total number of periodic and aperiodic tasks active in $[d', H + D')$, where d' is the earliest deadline in the current group, D' is the latest deadline of all pending aperiodic tasks at the time of the group admission and H is the hyper-period of the system. Tia in [31] proposed a technique to compute slack in dynamic priority schedules of periodic tasks requiring $\Theta(N^2 + N)$ storage overhead, which is impractical for all practical purposes. Silly in [5] presented an algorithm to schedule a single soft aperiodic task in time $O(n\frac{\max_i\{D_i\}}{\min_i\{T_i\}})$, where $\max_i\{D_i\}$ and $\min_i\{T_i\}$ are the maximum deadline and minimum period, respectively, among all periodic tasks. The storage complexity is $O(N)$, where N is the total number of periodic jobs released in one hyper-period. Other methods for feasibility testing include the methods by Ripoll *et al.*, [32], [33] and Buttazzo *et al.*, [34], [35] for dynamic priority systems. On-line feasibility algorithms based on their methods have pseudo-polynomial complexity in order to guarantee a single task.

Hohler [36] and Isović *et al.*[37], [38] have proposed the slot-switching method.

Several schedule construction methods have been developed in the past. Buttazzo *et al.*, in [39], [40] proposed the RED method, where tasks are serviced by the EDF discipline and feasibility testing is performed dynamically on every task in the system, periodic and aperiodic. The time complexity is $O(M)$ where M is the total number of tasks in the ready queue and it can be pseudo-polynomial.

III. Workload Theorems

Theorem 3 (Workload Dominance) Given a periodic task system $\boldsymbol{\tau}$, let $W_i^{\mathrm{F}}(t)$ and $W_i^{\mathrm{L}}(t)$ denote the serviced workload functions under the $\mathcal{S}^{\mathrm{F}}$ (FPF) and $\mathcal{S}^{\mathrm{L}}$ (LDL) schedules, respectively. The following holds

$$W_i^{\mathrm{F}}(t) \geq W_i^{\mathrm{L}}(t), \tag{23}$$

for all $i = 1, 2, \dots, n$ and $t > 0$.

Proof: From Eq. (6) and (11), for all $t \geq 0,\ i = 1, 2, \dots, n$, we obtain

$$W_i^{\mathrm{F}}(t) = \sum_{j=1}^{i}\big((n_j(t)-1)\cdot C_j + c_j^{\mathrm{F}}(t)\big),$$
$$W_i^{\mathrm{L}}(t) = \sum_{j=1}^{i}\big((n_j(t)-1)\cdot C_j + c_j^{\mathrm{L}}(t)\big),$$

where, $c_i^{\mathrm{F}}(t)$ and $c_i^{\mathrm{L}}(t)$, are the serviced workloads for τ_i in the currently active periods, under FPF and LDL, respectively. We will use mathematical induction on the number of periodic tasks.

Basis $[i = 1]$ For any $t \geq r_1$, let $t = r_1 + (j-1)T_1 + x$, where $j = n_1(t)$ and $x = m_1(t)$, $0 \leq x < T_1$, we have, $W_1^S(t) = (n_1(t)-1)\cdot C_1 + c_1^S(t)$, where $S = $ F or L and

$$c_1^{\mathrm{F}}(r_{1j}+x) = \min\big\{r_{1j}+x-r_{1j}, C_1\big\} = \min\big\{x, C_1\big\},\ \text{and,}$$
$$c_1^{\mathrm{L}}(r_{1j}+x) = C_1 - \min\big\{[r_{1j}+D_1-(r_{1j}+x)]^+, C_1\big\} = C_1 - \min\big\{D-x, C_1\big\}.$$

It is easy to verify that, for all r_{1j} and x, with $r_1 \leq x < T_1$, $c_1^{\mathrm{F}}(r_{1j}+x) \geq c_1^{\mathrm{L}}(r_{1j}+x)$, and thus, $c_1^{\mathrm{F}}(t) \geq c_1^{\mathrm{L}}(t)$ and $W_i^{\mathrm{F}}(t) \geq W_i^{\mathrm{L}}(t)$.

Inductive Hypothesis $[i = k-1]$ We assume that

$$W_i^{\mathrm{F}}(t) \geq W_i^{\mathrm{L}}(t)$$

for all $t \geq r_{k-1}$ and all $i = 1, \dots, k-1,\ k \leq n$. The only way the theorem can become false is when there is a point where the $W_k^{\mathrm{F}}(t)$ and $W_k^{\mathrm{F}}(t)$ meet and the latter one becomes larger than the former one. Without loss of generality, we assume the first point this happens is $y \in [r_{k,n_k(y)}, d_{k,n_k(y)})$ where $W_k^{\mathrm{F}}(y) = W_k^{\mathrm{L}}(y)$ and for which $W_k^{\mathrm{F}}(x) > W_k^{\mathrm{L}}(x)$, for all $x \in [r_{k,n_k(y)}, y)$ and $W_k^{\mathrm{F}}(x) < W_k^{\mathrm{L}}(x)$, for some $x > y$. Since both $W_k^{\mathrm{F}}(t)$ and $W_k^{\mathrm{L}}(t)$ are piecewise continuous, with line segments which either are parallel to the x-axis or increase with a slope of 45^o, it must be the case that for $t \in [y-\varepsilon, y+\varepsilon),\ \varepsilon > 0,\ W_k^{\mathrm{F}}(t)$ is parallel to the the x-axis while $W_k^{\mathrm{L}}(t)$ is increasing and they intersect at $t = y$. However, when $W_k^{\mathrm{F}}(t)$ is parallel to the the x-axis the unfinished workload $V_k(y) - W_k^{\mathrm{F}}(y) = 0$, and similarly, since $W_k^{\mathrm{L}}(t)$ is increasing at $t = y$ the unfinished workload $V_k(y) - W_k^{\mathrm{L}}(y) > 0$. This would imply that $W_k^{\mathrm{F}}(y) > W_k^{\mathrm{L}}(y)$, which is a contradiction of our assumption that $W_k^{\mathrm{F}}(y) = W_k^{\mathrm{L}}(y)$. Therefore, since the assumption that there is a point y after which $W_k^{\mathrm{F}}(y+\varepsilon) < W_k^{\mathrm{L}}(y+\varepsilon),\ \varepsilon > 0$ leads to a contradiction, we conclude that it is not possible that $W_k^{\mathrm{F}}(t) < W_k^{\mathrm{L}}(t)$. ■

Corollary 2: Let $Z_i^{\mathrm{F}}(t)$ and $Z_i^{\mathrm{L}}(t)$ denote the idle capacity functions on the $\mathcal{S}^{\mathrm{F}}$ and $\mathcal{S}^{\mathrm{L}}$ schedules, respectively. Then, for all $t \geq 0$ and all $i = 1, 2, \dots, n$, the following is true,

$$Z_i^{\mathrm{F}}(t) \leq Z_i^{\mathrm{L}}(t). \tag{24}$$

Proof: It follows immediately from Theorem 3. ■

Theorem 4 (Workload Switch) Given a periodic task system $\boldsymbol{\tau}$, assume that the system dispatches the periodic tasks in the interval $[0, a)$ under FPF, at a it starts dispatching the tasks by the LDL schedule and it continues to do so until time $t > a$. Then the serviced workload in $[a, t)$, denoted as $dW_n^{\mathrm{FL}}(a, t)$, is less than or equal to the workload serviced had the system remained under the FPF schedule, denoted by $dW_n^{\mathrm{FF}}(a, t)$, that is

$$dW_i^{\mathrm{FL}}(a, t) \leq dW_i^{\mathrm{FF}}(a, t) \tag{25}$$

for all $i = 1, 2, \dots, n$ and $t \geq a \geq 0$.

Proof: We have that

$$dW_i^{\mathrm{FF}}(a,t) = \sum_{k=1}^{i} \left[(n_k(t) - n_k(a))C_k + c_k^{\mathrm{F}}(t) - c_k^{\mathrm{F}}(a)\right]^+$$
$$dW_i^{\mathrm{FL}}(a,t) = \sum_{k=1}^{i} \left[(n_k(t) - n_k(a))C_k + c_k^{\mathrm{L}}(t) - c_k^{\mathrm{F}}(a)\right]^+$$

where, by Theorem 3, $c_k^{\mathrm{F}}(t) \geq c_k^{\mathrm{L}}(t)$. Therefore, $dW_i^{\mathrm{FL}}(a,t) \leq dW_i^{\mathrm{FF}}(a,t)$. ■

IV. Experimental Task Sets

In this part of the paper we give details on the periodic task sets we have used in the examples and for the experiments. Each table shows the number of periodic tasks n, the hyper-period H, the utilization ρ and the discretization unit q. Sets TS1, TS2 and TS3 are synthetic and T1, T2, T3, T4, T5, T6, T7 and T71 are actual real-time task sets.

TABLE III
Synthetic Task Set TS1

n	H	ρ	q
5	$10sec$	0.75	0.01
C_i	T_i	D_i	r_i
0.020	0.100	0.100	0.000
0.030	0.200	0.200	0.000
0.100	1.000	1.000	0.000
0.500	5.000	5.000	0.000
2.000	10.000	10.000	0.000

TABLE IV
Synthetic Task Set TS2

n	H	ρ	q
5	$77sec$	0.356494	0.01
C_i	T_i	D_i	r_i
0.01	0.1	0.1	0.0
0.03	0.7	0.7	0.0
0.09	2.2	2.2	0.0
0.40	5.5	5.5	0.0
1.10	11.0	11.0	0.0

TABLE V
Synthetic Task Set TS3

n	H	ρ	q
10	$231sec$	0.860823	0.01
C_i	T_i	D_i	r_i
0.01	0.1	0.1	0.0
0.01	0.2	0.2	0.0
0.02	0.7	0.7	0.0
0.03	0.7	0.7	0.0
0.12	2.1	2.0	0.0
0.14	2.1	2.0	0.0
0.22	4.2	4.1	0.0
1.22	4.2	4.1	0.0
0.40	5.5	5.5	0.0
1.10	11.0	11.0	0.0

TABLE VI
Inertial Navigation Task Set T1

n	H	ρ	q
6	$5000sec$	0.884040	0.01
C_i	T_i	D_i	r_i
1.18	2.500	2.500	0.000
4.28	40.000	40.000	0.000
10.28	62.500	62.500	0.000
20.28	1000.000	1000.000	0.000
100.28	1000.000	1000.000	0.000
25.00	1250.000	1250.000	0.000

TABLE VII
Submarine Sonar Task Set T2

n	H	ρ	q
6	$60000sec$	0.572500	1
C_i	T_i	D_i	r_i
50.0	100.0	100.0	0.0
9.0	500.0	500.0	0.0
41.0	1000.0	1000.0	0.0
5.0	2500.0	2500.0	0.0
33.0	3000.0	3000.0	0.0
2.0	4000.0	4000.0	0.0

TABLE VIII
Surface Ship Radar Task Set T3

n	H	ρ	q
11	$20000sec$	0.837500	5
C_i	T_i	D_i	r_i
25.0	2000.0	2000.0	0.0
10.0	4000.0	4000.0	0.0
25.0	5000.0	5000.0	0.0
35.0	5000.0	5000.0	0.0
400.0	5000.0	5000.0	0.0
25.0	10000.0	10000.0	0.0
30.0	10000.0	10000.0	0.0
50.0	10000.0	10000.0	0.0
300.0	10000.0	10000.0	0.0
1125.0	10000.0	10000.0	0.0
5775.0	10000.0	10000.0	0.0

TABLE IX
Avionics System Task Set T4

n	H	ρ	q
15	$118000sec$	0.830093	1
C_i	T_i	D_i	r_i
2.0	25.0	25.0	0.0
5.0	25.0	25.0	0.0
1.0	40.0	40.0	0.0
3.0	50.0	50.0	0.0
5.0	50.0	50.0	0.0
8.0	59.0	59.0	0.0
9.0	80.0	80.0	0.0
2.0	80.0	80.0	0.0
5.0	100.0	100.0	0.0
3.0	200.0	200.0	0.0
1.0	200.0	200.0	0.0
1.0	200.0	200.0	0.0
3.0	200.0	200.0	0.0
1.0	1000.0	1000.0	0.0
1.0	1000.0	1000.0	0.0

TABLE X
Multimedia System Task Set T5

n	H	ρ	q
11	$300000sec$	0.684450	1
C_i	T_i	D_i	r_i
28.0	125.0	125.0	0.0
19.0	250.0	250.0	0.0
1175.0	6000.0	6000.0	0.0
9.0	12000.0	12000.0	0.0
1880.0	25000.0	25000.0	0.0
1880.0	30000.0	30000.0	0.0
5000.0	100000.0	100000.0	0.0

TABLE XI
The Satellite Control System Task Set T6, without the Sporadic Tasks and with All $r_i = 0$

n	H	ρ	q
10	$1sec$	0.579220	0.000010
C_i	T_i	D_i	r_i
0.00028	0.050	0.009	0.0
0.00176	0.010	0.010	0.0
0.00213	0.200	0.014	0.0
0.00143	0.200	0.017	0.0
0.00143	0.200	0.022	0.0
0.00143	0.100	0.024	0.0
0.00821	0.100	0.050	0.0
0.05284	0.200	0.200	0.0
0.00516	1.000	0.400	0.0
0.00691	1.000	0.900	0.0

TABLE XII
The Satellite Control System Task Set T7, without the Sporadic Tasks

n	H	ρ	q
10	$1sec$	0.579220	0.000010
C_i	T_i	D_i	r_i
0.00028	0.050	0.009	0.00
0.00176	0.010	0.010	0.00
0.00213	0.200	0.014	0.05
0.00143	0.200	0.017	0.05
0.00143	0.200	0.022	0.00
0.00143	0.100	0.024	0.00
0.00821	0.100	0.050	0.05
0.05284	0.200	0.200	0.05
0.00516	1.000	0.400	0.20
0.00691	1.000	0.900	0.20

TABLE XIII
The Satellite Control System Task Set T71, with the Sporadic Tasks

n	H	ρ	q
13	$3sec$	0.858560	0.000010
C_i	T_i	D_i	r_i
0.00018	0.00096	0.00063	0.00
0.00028	0.050	0.009	0.00
0.00176	0.010	0.010	0.00
0.00213	0.200	0.014	0.05
0.00143	0.200	0.017	0.05
0.00143	0.200	0.022	0.00
0.00143	0.100	0.024	0.00
0.00319	0.0625	0.030	0.00
0.00821	0.100	0.050	0.05
0.00408	0.100	0.050	0.05
0.05284	0.200	0.200	0.05
0.00516	1.000	0.400	0.20
0.00691	1.000	0.900	0.20