The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments

Jay K. Strosnider, *Member, IEEE*, John P. Lehoczky, *Member, IEEE*, and Lui Sha, *Senior Member, IEEE*

***Abstract*— Most existing scheduling algorithms for hard real-time systems apply either to periodic tasks or aperiodic tasks but not to both. In practice, real-time systems require an integrated, consistent approach to scheduling that is able to simultaneously meet the timing requirements of hard deadline periodic tasks, hard deadline aperiodic (alert-class) tasks, and soft deadline aperiodic tasks. This paper introduces the Deferrable Server (DS) algorithm which will be shown to provide improved aperiodic response time performance over traditional background and polling approaches. Taking advantage of the fact that, typically, there is no benefit in early completion of the periodic tasks, the Deferrable Server (DS) algorithm assigns higher priority to the aperiodic tasks *up until the point* where the periodic tasks would start to miss their deadlines. Guaranteed alert-class aperiodic service and greatly reduced response times for soft deadline aperiodic tasks are important features of the DS algorithm, and both are obtained with the hard deadlines of the periodic tasks still being guaranteed. The results of a simulation study performed to evaluate the response time performance of the new algorithm against traditional background and polling approaches are presented. In all cases, the response times of aperiodic tasks are significantly reduced (often by an order of magnitude) while still maintaining guaranteed periodic task deadlines.**

***Index Terms*— Aperiodics, hard deadlines, deferrable server, periodics, real-time, response times, schedulability.**

I. Introduction

MOST existing scheduling algorithms for hard real-time systems apply either to periodic tasks or aperiodic tasks but not both. In practice, however, most real-time systems require an integrated, consistent approach suitable for scheduling hard deadline periodic tasks along with both hard and soft deadline aperiodic tasks. The periodic tasks typically arise from sensor data or control loops, while the aperiodic tasks generally arise from operator actions or aperiodic events. Most aperiodic tasks have average response time requirements (soft deadlines). However, some aperiodic tasks, such as alerts, may require guaranteed response times. We define alerts as aperiodic tasks with very high semantic importance requiring guaranteed response time performance. The problem is to jointly schedule the periodic and aperiodic tasks so that the individual timing requirements for all tasks are met. In particular, the hard deadlines of periodic tasks and aperiodic alerts must be met, and the average response times for the other aperiodic tasks should be minimized.

One commonly used approach is to treat the aperiodic tasks as background tasks when their response times are not critical. A second commonly used approach when timing requirements are more stringent is to use polling or time division multiplexing (TDM) schemes. This paper presents a new scheduling algorithm which is designed to offer fast response time performance for aperiodic tasks while still guaranteeing the hard deadlines of periodic tasks at a high level of periodic tasks utilization. The new algorithm, called the Deferrable Server (DS) algorithm, is able to substantially reduce the response time of aperiodic tasks by delaying the completion time of periodic tasks while still ensuring that their deadlines are met. The DS algorithm is built upon the rate monotonic scheduling algorithm [2] which features an attractive combination of high performance, predictable behavior and ease of implementation.

Under the assumption of preemptive scheduling and task deadlines equal to task periods, Liu and Layland [2] proved that the rate monotonic algorithm is the optimal fixed priority scheduling algorithm. Here, optimality means that if a fixed priority scheduling algorithm can meet all deadlines of any periodic task set, then so can the rate monotonic algorithm. Fixed priority scheduling algorithms cannot always achieve 100% processor utilization and still meet all task deadlines. Liu and Layland derived a least upper bound on processor utilization, given by $n(2^{1/n} - 1)$ for a task set with n periodic tasks. Any task set with total processor utilization at or below this bound could be scheduled by the rate monotonic algorithm. Joseph and Pandya [14] and Lehoczky, Sha, and Ding [6] later developed a necessary and sufficient (exact) schedulability tests that can be used to determine the schedulability of any given periodic task set. Recent work by Lehoczky [1] expanded the exact scheduling criterion to allow for general task deadlines. In addition, the rate monotonic algorithm has been greatly generalized to incorporate effects such as

Manuscript received July 19, 1989; revised May 7, 1990, October 12, 1991, March 1, 1992. This work was supported in part from a grant from the Office of Naval Research and in part from a grant from Naval Ocean Systems Center.

J. K. Strosnider is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA; E-mail: strosnider@ece.cmu.edu.

J. P. Lehoczky is with the Department of Statistics, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

L. Sha is with the Software Engineering Institute and School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

IEEE Log Number 9407113.

task synchronization and transient overload. A comprehensive summary can be found in Lehoczky, Sha, Strosnider and Tokuda [7].

Other related scheduling work includes work by Liu, Liu and Liestman [3] who developed slack time bounds for periodic tasks scheduled using the rate monotonic algorithm. Leung and Merrill [4], and Lawler [5] considered deadline scheduling on multiple processors. Mok [8] showed that the least slack-time algorithm is an optimal algorithm for scheduling preemptable periodic tasks on a single processor, and also proved that the least slack-time algorithm dominates the earliest deadline scheduling algorithm on multiprocessor scheduling. Sha [9] showed that the dynamic algorithms are unstable under transient overload, meaning that under certain circumstances the deadlines of tasks can be missed in an unpredictable manner. In contrast, the transient overload problem can be easily solved in the context of the rate monotonic scheduling algorithm [9]. Leung [4] proved that when the start time and the deadline for a task do not coincide with the period boundaries of a periodic task, that it is an NP-hard problem to decide if such a periodic task set can be scheduled by the earliest deadline algorithm [4]. Mok [8] found the same difficulty when tasks use semaphores for mutual exclusion.

This paper addresses a new aspect of hard real-time scheduling, namely the joint scheduling of hard deadline periodic tasks and aperiodic tasks. Specifically, the approach taken is the construction of a special periodic server task for processing aperiodic tasks. If the server task's capacity is available only at periodic instants, then it is the same as a polling task (hereafter called a Polling Server). By contrast, the Deferrable Server or DS capacity is available for processing aperiodic tasks arriving at any time in its period, a modification which leads to better aperiodic task response times. The inclusion of a server task, whether a Polling Server or a Deferrable Server task calls for a new analysis for two distinct reasons. First, the utilization of the Polling and DS Server tasks will be fixed, consequently, the Liu and Layland least upper bound can be increased by taking this utilization explicitly into account. Second, the DS task violates one of the Liu and Layland assumptions, namely that the task is ready at the start of the task period. Consequently, a modification of Liu and Layland's analysis must be carried out. These new analyses are important contributions of this paper.

This paper is organized as follows. Section II sets the context for the aperiodic scheduling problem and introduces the Deferrable Server algorithm. Section III extends the Lehoczky, Sha and Ding exact case schedulability analysis [6] to include a Deferrable Server task as the highest priority task. Section IV first develops least upper bounds for the two tasks case for both the Polling Server and the Deferrable Server, and then summarizes the least upper bounds for the general case. Appendixes A and B provide proofs for the Polling Server and Deferrable Server least upper bounds respectively. Section IV-C provides guidelines for designing Deferrable Servers. Section V summarizes the results of simulation studies run to evaluate the response time performance of the algorithms. Section VI provides a summary and conclusions.

II. Deferrable Server DS Algorithm Description

The DS algorithm extends the rate monotonic scheduling algorithm to provide abstractions to support the scheduling of aperiodic tasks requiring quick response times. The rate monotonic algorithms for scheduling periodic tasks was formalized by Liu and Layland [2] under the following assumptions:

- A.1) All periodic tasks, τ, have periods, T, and constant, known execution times, C. Further, tasks are ready for execution at the beginning of each period,
- A.2) Task deadlines, D, are at the end of the task periods, that is $D = T$.
- A.3) Tasks are independent, do not synchronize or block each other and do not suspend themselves.
- A.4) All overhead for scheduling, context swapping, etc., is assumed to be zero.

The rate monotonic algorithm assigns priorities in inverse relation to task periods, that is the shorter the task's period, the higher the task's priority with ties broken arbitrarily. Throughout the following analysis, we first analyze the Polling Server PS and then extend the analysis to address the DS case.

PS Algorithm Description: A Polling Server is a periodic task with period T_{PS} and an execution time C_{PS}. The PS is used to provide relatively high priority service to aperiodic task arrivals. It is ready to run at the start of its period and services pending or arriving aperiodic tasks over the interval from the start of its period until C_{PS} time units later. The PS task is subject to preemption by higher priority tasks, until either it exhausts its execution time or there is no aperiodic work left to be executed. In the latter case, it loses any of the unused execution time and is unavailable to service aperiodic tasks until the start of its next period. The PS task is scheduled as if it were a periodic task with period T_{PS}. Aperiodic tasks that arrive or remain when the PS is unavailable can be serviced at background priority.

DS Algorithm Description: A Deferrable Server (DS) is a periodic task with period T_{DS} and capacity C_{DS}. The DS is used to provide high priority service to aperiodic tasks. It is ready at the start of its period and services aperiodic task arrivals, subject to preemption by higher priority tasks, until it exhausts its execution time, C_{DS}, or the end of its period is reached. Unlike the PS which loses any unused execution time when there is no aperiodic work remaining, the DS execution time, C_{DS}, is available for servicing aperiodic arrivals throughout its entire period. It loses any unused execution time at the end of its period when its full capacity C_{DS} is restored. The DS task is scheduled as if it were a periodic task with period T_{DS}. Aperiodic tasks that arrive when the DS execution time, C_{DS}, has been exhausted can be serviced at background priority.

In general, the DS task is assigned a priority according to the rate monotonic algorithm based on its period, T_{DS}, relative to the other periodic tasks. While the DS task can execute at any priority level, assigning the DS task the highest priority (by giving it a period no longer than the shortest periodic task period) allows one to guarantee that the deadlines of aperiodic alerts are met as well as enhancing the respon-

siveness of the soft deadline aperiodic tasks. At intermediate priority levels, the DS is less capable of providing responsive aperiodic service. Moreover, DS capacity can be lost because of higher priority preemptions even when aperiodics are ready for processing. For these reasons, we only consider the DS at the highest priority level.

The DS task is different from the other periodic tasks. Each of the other tasks correspond to specific periodic tasks which are assumed to be ready to run at the start of their respective periods. The DS task is demand driven and can run in any part of its period in response to aperiodic arrivals. This characteristic, along with the fact that the DS task typically runs at the highest priority, provides highly responsive service for aperiodic tasks in hard real-time environments. On the other hand, the ability of the DS task to defer its execution time until later in its period when it may be needed violates assumption A1. The schedulability analysis developed by Liu and Layland [2] no longer applies, and a new analysis must be developed.

III. Necessary and Sufficient Schedulability Conditions

We next present an analysis leading to a determination of the schedulability of a DS task, τ_0, executing at the highest priority together with m additional periodic tasks, $\tau_1, \cdots, \tau_m$, with priority assignments given by the rate monotonic algorithm. Each of the $m+1$ tasks is characterized by a period, T_i, an execution time, C_i, and a phasing relative to 0, I_i, satisfying $0 \leq I_i < T_i$. We assume that $T_0 \leq T_1 \leq T_2 \leq \cdots \leq T_m$, and assign priorities consistent with the rate monotonic scheduling algorithm. The m ordinary periodic tasks have deadlines equal to their period. The kth execution request of τ_i, $1 \leq i \leq m$ is ready at $I_i + (k-1)T_i$, and executes for C_i units of time by its deadline at $I_i + kT_i$, $k \geq 1$. The DS task, τ_0 provides C_0 units of available execution time starting at $I_0 + (k-1)T_0$. This execution time is available for use throughout the interval $[I_0 + (k-1)T_0, I_0 + kT_0)$, and any unused capacity is lost at $I_0 + kT_0$. We will discuss the choices of C_0 and T_0 to achieve good aperiodic response and minimize the wasted capacity in Section IV-C. We first focus on determining criteria that ensure that all deadlines of the periodic tasks are guaranteed.

In this section, we derive necessary and sufficient conditions which will ensure that all deadlines of the periodic tasks are always met for any task phasing. The derivation of the necessary and sufficient schedulability conditions relies upon the following three results from Liu and Layland [2], and Lehoczky, Sha and Ding [6]:

Result 1: A task τ_i has its longest response time when it arrives at a *critical instant*. A critical instant occurs at $t = 0$ when $I_i = 0$, $1 \leq i \leq m$.

Result 2: All task deadlines will be met using the rate monotonic scheduling algorithm if the first request for each task meets its deadline under critical instant phasing, $I_i = 0$, $1 \leq i \leq m$.

Result 3: All periodic task deadlines are guaranteed by the rate monotonic algorithm under all task phasings if and only if

$$\max_{1 \leq i \leq m} \min_{0 \leq t \leq T_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \leq 1. \tag{1}$$

The third result, which is the necessary and sufficient schedulability condition as stated in [6], relied on the earlier two results from Liu and Layland, and will be used to determine the schedulability of task sets which include a PS. In this paper we will generalize the above three results to permit inclusion of the DS task τ_0 at the highest priority level.

We define the response time of a task to be the time between its arrival and its completion. We want to find the conditions for the longest response time for any execution request of task $\tau_i, 1 \leq i \leq m$. To do this, we introduce the concept of a level-i busy period on the processing resource.

Definition: A level-i busy period is a time interval $[s, t]$ satisfying the following three conditions:

1) All requests of priority i or higher made before s are completed by s,
2) All requests of priority i or higher made before t are completed by t.
3) For every $u \in (s, t)$, there exists at least one request of priority i or higher that arrived before u and is not completed by u.

To illustrate this concept, consider a task set consisting of two tasks: $\tau_1 : C_1 = 4$, $T_1 = 6$, $I_1 = 0$ and $\tau_2 : C_2 = 6$, $T_2 = 14$, $I_2 = 0$, with τ_i having priority i, $i = 1, 2$. During [0, 24] the level-1 busy periods are [0, 4], [10, 14] and [20, 24], while the level-2 busy periods are [0, 10], [10, 20] and [20, 24]. Note that the processor is continuously busy with tasks of priority 2 or higher throughout [0, 24]; however, the level-2 busy periods are separated by the time points 10 and 20 at which all outstanding requests have been completed. The above example shows that a level-i busy period may or may not contain a request for τ_i. If, as we have assumed, $D_i \leq T_i$, all deadlines are met and a level-i busy period contains a request for τ_i, then that busy period will contain exactly one such request, and it will end when that request is completed. This follows because no task of priority higher than i can be active when τ_i completes, and at most one request for τ_i can be active if all deadlines are met.

The concept of level-i busy period can be used to characterize the longest response time of any request for τ_i.

Lemma 3.1: Suppose a periodic task set $\tau_1, \cdots, \tau_n$ is schedulable using a fixed priority scheduling algorithm where τ_i is assigned priority i. A necessary condition for an execution request of τ_i to have its longest response time is for the request of τ_i to initiate a level-i busy period.

Proof: Tasks $\tau_{i+1}, \cdots, \tau_n$ are of lower priority than τ_i and can be preempted by τ_i, thus they can be ignored. Consider any phasing of tasks $\tau_1, \tau_2, \cdots, \tau_i$. Suppose that the execution request of τ_i does not initiate a level-i busy period. This means that τ_i's request must occur after a level-i busy period started. Let the start of the level-i busy period be $s < t_i$, where t_i denotes the time of τ_i's request. During $[s, t_i]$, the processor is continuously busy with work of priority higher than i. For the given phasing, τ_i will complete at some time $t > t_i$, giving

a response time of $t - t_i$. Thus there are exactly C_i units of computation available to τ_i in $[t_i, t]$, and none in $[s, t_i]$. If τ_i had been initiated at time s, then it would also have completed at time t, since there is no level-i computation time available to it in $[s, t_i]$. This would increase the response time to $s - t$. This shows that the response time of task τ_i is maximized only if its request initiates a level-i busy period. □

We next find the worst case phasing of the tasks, that is the phasing that maximizes the response time of task i.

Theorem 3.2: The task set phasing which causes the longest response time for any periodic task occurs when all task periods of equal or higher priority are requested simultaneously, and the highest priority Deferrable Server task demands C_0 units at this instant and is reinitiated C_0 units later. To check whether a task set is schedulable using a fixed priority algorithm with the Deferrable Server task having the highest priority, one need only check whether the first execution request of each task meets its deadline under this worst case phasing.

Proof: Using lemma 3.1, we may assume that τ_i's request initiates a level-i busy period, so at the time of its request, which we refer to as 0, there is no accumulated work of higher priority. Hence, τ_i's period starts at 0. All other periods of tasks with priority higher than i start at I_j, $1 \le j < i$ with $I_j \ge 0$. The worst case I_j occurs when the processor must devote the largest amount of time to τ_j during $[0, t]$, where t is the completion time of τ_i. Given I_j, the cumulative demand for the processor made by any non-DS periodic task τ_j in $[0, t]$ is given by

$$C_j \left\lceil \frac{\max(0, t - I_j)}{T_j} \right\rceil \tag{2}$$

which is maximized for all $t \ge 0$ by setting $I_j = 0$. The special case of the Deferrable Server task τ_0 is somewhat different. It can request C_0 units anywhere in any T_0 time units. The largest demand for processor time is attained by setting $I_0 = C_0$ and having execution requests in $[0, C_0], [C_0, 2C_0], [C_0 + T_0, 2C_0 + T_0]$ etc. Since the worst case phasing results in the longest response time for any task, if the first execution request meets its deadline under the worst case phasing, so will all subsequent execution requests. □

We can now generalize (1) to provide a necessary and sufficient schedulability test with a high priority DS task included. The analysis closely follows that presented in [6]. A task τ_k, $1 \le k \le n$ will meet all its deadlines under all phasings if its first execution request meets its deadline under the worst case phasing. The first execution request for τ_k will meet its deadline, if and only if there exists a time t before task τ_k's deadline at which C_k units of work for τ_k and all work of priority higher than k is completed. Under the worst case phasing, the total demand for processor time at time $t \ge T_0$ is given by $C_0 + C_0 \left\lceil \frac{t - T_0}{T_0} \right\rceil + \sum_{j=1}^{k} C_j \left\lceil \frac{t}{T_j} \right\rceil$, where the first two terms correspond to the maximum possible execution time of τ_0. Thus τ_k will meet its deadline if

$$\min_{T_0 \le t \le T_k} \left\{ C_0 \left(1 + \left\lceil \frac{t - T_0}{T_0} \right\rceil \right) + \sum_{j=1}^{k} C_j \left\lceil \frac{t}{T_j} \right\rceil \right\} \le t$$

or equivalently

$$\min_{T_0 \le t \le T_k} \left\{ \frac{1}{t} \left(C_0 \left(1 + \left\lceil \frac{t - T_0}{T_0} \right\rceil \right) + \sum_{j=1}^{k} C_j \left\lceil \frac{t}{T_j} \right\rceil \right) \right\} \le 1. \tag{3}$$

For the entire task set to be schedulable, (3) must hold for each k, $1 \le k \le m$. This leads to the necessary and sufficient schedulability condition given by

$$\max_{1 \le k \le n} \min_{T_0 \le t \le T_k} \left\{ \left(C_0 \left(1 + \left\lceil \frac{t - T_0}{T_0} \right\rceil \right) + \sum_{j=1}^{k} C_j \left\lceil \frac{t}{T_j} \right\rceil \right) / t \right\} \le 1. \tag{4}$$

The minimizations in (1), (3), and (4) call for finding the minimum of an expression with respect to a continuous variable t. The ceiling functions are step functions, consequently the terms in the expression are piecewise continuous and decreasing functions. This means that the formulas need to be evaluated only at points where the expression is discontinuous, that is the times t which are multiples of T_i for all $1 \le i \le k$. One fairly efficient method to determine schedulability of a task set is to compute the sequence of time points $\{S_\ell, \ell = 0, 1, \cdots\}$ with

$$S_0 = 2C_0 + \sum_{j=1}^{k} C_j \left\lceil \frac{T_0}{T_j} \right\rceil,$$

$$\text{and } S_{\ell+1} = C_0 \left(1 + \left\lceil \frac{S_\ell - T_0}{T_0} \right\rceil \right) + \sum_{j=1}^{k} C_j \left\lceil \frac{S_\ell}{T_i} \right\rceil.$$

If there exists an $\ell \ge 1$ such that $S_\ell = S_{\ell+1} \le T_k$, then τ_k is schedulable. If there exists $\ell \ge 1$ such that $S_\ell > T_k$, then τ_k is not schedulable. This check must be carried out for $k = 1, 2, \cdots, n$.

IV. Least Upper Bounds on Schedulability

In this section, we develop least upper bounds on task set schedulability generalizing the results of Liu and Layland [2] to the case of task sets that contain a high priority server task, a PS task or a DS task. The least upper bounds provide sufficient conditions for task set schedulability in the sense that if the task set utilization lies below the bound, all periodic task deadlines will be met. If, however, the utilization lies above the bound, then the necessary and sufficient schedulability tests given in (4) must be used to determine whether the task set is schedulable.

Given that the PS and DS tasks behave differently from the hard deadline periodic tasks, it is useful to keep the server task (DS or PS) utilization $U_0 = C_0/T_0$ distinct from the total periodic task utilization $U_{\text{per}} = U_1 + \cdots + U_n$ where $U_i = C_i/T_i$. Suppose that U_0 is fixed. We wish to find a

least upper bound on U_{per} denoted by $\text{PS}_n(U_0)$ and $\text{DS}_n(U_0)$ respectively for any given utilization U_0 for the PS or DS task. Any periodic task set with utilization less than or equal to $\text{PS}_n(U_0)$ or $\text{DS}_n(U_0)$ is schedulable in the presence of a high priority server task having utilization U_0. The bound is a least upper bound in the sense that for every $U_{\text{per}} > \text{PS}_n(U_0)$ or $\text{DS}_n(U_0)$ there exists a nonschedulable task set having utilization U_{per}. We will determine these least upper bounds for both high priority Polling Servers and Deferrable Servers.

A. The Case of Two Periodic Tasks

To illustrate the ideas, we begin with the special case of two periodic tasks. Consider the situation with no DS task and two hard deadline periodic tasks τ_0 and τ_1. The high priority periodic task could be a Polling Server. Note that the analysis for the PS is identical to the case of determining the new least upper bound given that the utilization of the highest priority periodic task is fixed. The resulting bound must in all cases be equal to or greater than the Liu and Layland least upper bound. Assume that C_0 and T_0 and T_1 are given. Let $R_1 = T_1/T_0 \geq 1$. We ask how large C_1 can be and still have all task deadlines satisfied under all phasings of τ_0 and τ_1. Assuming the worst case phasing, the maximum utilization for $U_1 = C_1/T_1 = C_1/R_1T_0$ is given by

$$\text{PS}_{U_1}(U_0, R_1) = \begin{cases} \dfrac{k(1-U_0)}{R_1} & \text{if} \quad k \leq R_1 \leq k + U_0 \\ 1 - \dfrac{(k+1)U_0}{R_1} & \text{if} \quad k + U_0 \leq R_1 \leq k+1 \end{cases} \tag{5}$$

for $k = 1, 2, \cdots$. For any given value of U_0, the maximum value of U_1 varies from a maximum of $1-U_0$ when $R_1 = 1$ to a minimum of $\frac{1-U_0}{1+U_0}$ where $R_1 = 1+U_0$. The least upper bound on U_1 is therefore given by $(1-U_0)/(1+U_0), 0 \leq U_0 \leq 1$. The total schedulable utilization is minimized when $U_0 = \sqrt{2} - 1$. For this value of U_0, the total schedulable utilization becomes $2(\sqrt{(2} - 1) = 0.828$, the Liu and Layland bound for two tasks. However, when U_0, is fixed, the least upper bound on schedulable utilization, $\text{PS}_{\text{tot}}(U_0)$, is given by

$$\text{PS}_{\text{tot}}(U_0) = (1 + U_0^2)/(1 + U_0), \quad 0 \leq U_0 \leq 1. \tag{6}$$

This two-task least upper bound is plotted in Fig. 1 as a function of the utilization of the high priority task U_0, along with the corresponding DS two-task least upper derived below.

The same type of analysis can be carried out for a high priority DS task, τ_0, and one periodic task, τ_1. Again we assume C_0, T_0, and T_1 are specified, and we seek the largest value of C_1 for which all deadlines of τ_1 are met for all task phasings. The largest value of C_1 depends upon C_0, T_0 and R_1 and is given by

$$C_1 = \begin{cases} T_1 - 2C_0 & \text{if } 1 \leq R_1 \leq 1 + U_0 \\ k(T_0 - C_0) & \text{if } k + U_0 \leq R_1 \leq k + 2U_0, \quad k \geq 1 \\ T_1 - (k+2)C_0 & \text{if } k + 2U_0 \leq R_1 \leq k + 1 + U_0, \quad k \geq 1. \end{cases} \tag{7}$$

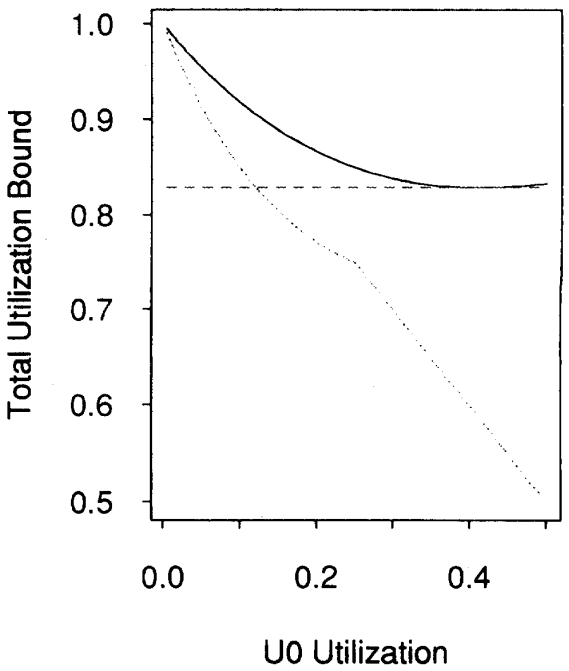


Fig. 1. Two-task least upper scheduling bounds. Solid: polling; dotted: DS; dashed: L&L.

Converting (7) to the maximum schedulable utilization of τ_1, we find it to be

$$\text{DS}_{U_1}(U_0, R_1) = \begin{cases} 1 - \dfrac{2}{R_1}U_0 & \text{if} \quad 1 \leq R_1 \leq 1 + U_0 \\ \dfrac{k(1-U_0)}{R_1} & \text{if} \quad k + U_0 \leq R_1 \leq k + 2U_0, \quad k \geq 1 \\ 1 - \frac{k+2U_0}{R_1} & \text{if} \quad k + 2U_0 \leq R_1 \leq k + 1 + U_0, \quad k \geq 1. \end{cases} \tag{8}$$

The above formulas provide least upper schedulability bounds for one lower priority periodic task as a function of U_0, the DS utilization, and $R_1 = T_1/T_0$. We can also find the value of R_1 which minimizes U_1. Minimizing (8) over R_1 we find

$$\text{DS}_{U_1}(U_0) = \begin{cases} \min(1-2U_0, (1-U_0)/(1+2U_0)) = B_1(U_0) & 0 \leq U_0 \leq 0.5 \\ 0 & 0.5 \leq U_0 \leq 1 \end{cases}. \tag{9}$$

The resulting least upper bound on total utilization for the two-task DS case is

$$\text{DS}_{\text{tot}}(U_0) = \begin{cases} \min(1 - U_0, (1 + 2U_0^2)/(1 + 2U_0)) & \text{if} \quad 0 \leq U_0 \leq 0.5 \\ U_0 & \text{if} \quad 0.5 \leq U_0 \leq 1 \end{cases}. \tag{10}$$

This least upper bound is plotted in Fig. 1 along with the corresponding PS least upper bound, and the Liu and Layland least upper bound. We only plotted the bound for the range

$0 \leq U_0 \leq 0.5$. Above this range, no periodic task τ_1 is schedulable, and the entire processor becomes devoted to aperiodic task processing.

The decrease in total schedulable utilization that occurs when a DS task with utilization U_0 replaces an ordinary periodic task with utilization U_0 for $0 \leq U_0 \leq 1/4$ is given by $\frac{1-U_0}{1+U_0} - \frac{1-U_0}{1+2U_0} = \frac{(1-U_0)U_0}{(1+U_0)(1+2U_0)}$. This is an increasing function of U_0. It is 0 when $U_0 = 0$ and increases to .1 when $U_0 = 1/4$. For $U_0 > 1/4$ a different least upper bound applies to the DS case, and a substantial decrease in utilization occurs. As will be seen in Section 5, the decrease in schedulable utilization can often pay for itself in the improved aperiodic task response time performance provided by the DS task.

B. The General Case

Polling Server Bounds: We now turn to the general case when there is an arbitrary number of periodic tasks. We first consider the case of no DS server task but $n+1$ periodic tasks where the highest priority task, τ_0, has fixed utilization of U_0. This highest priority task could be a PS task. We will later replace this task with a DS task.

Although the development of the least upper bounds for the PS are general for a PS running at any priority, we limit our discussion to the case where the PS is the highest priority task which is the case that will later be compared to the DS which is the primary focus of this paper. We consider a task set $\tau_0, \cdots, \tau_n$ where τ_0 is the PS task with a utilization U_0. We seek to find the least upper schedulability bound, $\mathrm{PS}_n(U_0)$ such that if the task set has utilization no greater than $\mathrm{PS}_n(U_0)$, then it is schedulable using the rate monotonic algorithm. However, for every utilization $U > \mathrm{PS}_n(U_0)$ there exists an unschedulable task set with a task having utilization U_0 and the task set having total utilization U.

The development of the least upper scheduling bounds for the general case for periodic task sets with a PS task is similar in form to the two-task case developed earlier but much more complicated. As such, we limit our in-line discussion to a summary of the PS bounds and provide the formal derivations in Appendix A. Appendix A proves that the least upper schedulability bound for total periodic task set utilization with a fixed PS task utilization of U_0 to be

$$\mathrm{PS}_{\mathrm{tot},n}(U_0) = n\left(\left(\frac{2}{1+U_0}\right)^{1/n} - 1\right) + U_0. \qquad (11)$$

Equation (11) has two terms, the first gives the utilization of the periodic tasks, while the second corresponds to the utilization of the Polling Server. Letting $n \to \infty$, one can find the limiting value for the least upper bound,

$$\mathrm{PS}_{\mathrm{tot},\infty}(U_0) = \ell n\left(\frac{2}{1+U_0}\right) + U_0. \qquad (12)$$

Equation (12) is a generalization of the Liu and Layland bound of $\ell n2$ in which the highest priority task has a utilization fixed to be U_0. A graph of this bound is given in Fig. 2. One can see that there is very little change until U_0 becomes very large. For example, when $U_0 = 0.25, \mathrm{PS}_{t,\infty}(U_0) = 0.720$, only slightly larger than $\ell n2 = 0.693$, the usual Liu and Layland bound.

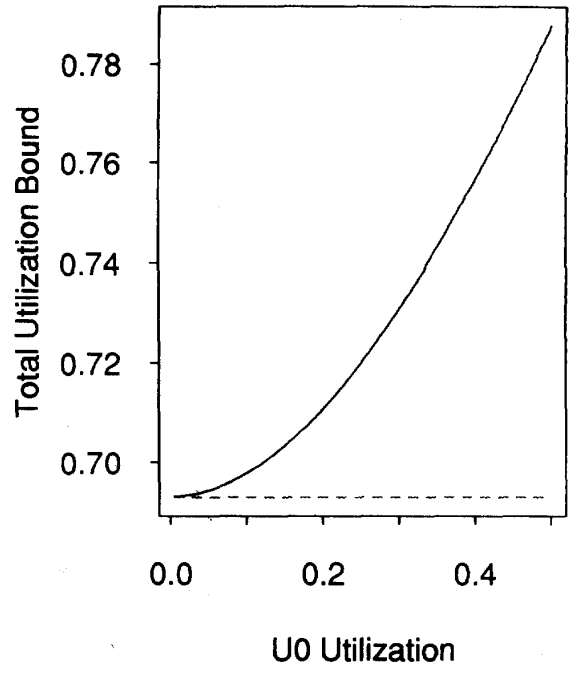


Fig. 2. Polling server least upper scheduling bound. Solid: polling; dashed: L&L.

Equation (11) shows that if one has a polling task with utilization U_0, then all periodic task sets consisting of n standard periodic tasks (in addition to the polling task) with total utilization no greater than $n\left(\left(\frac{2}{1+U_0}\right)^{1/n} - 1\right)$ will be schedulable. We now wish to determine a similar expression when the polling task is replaced by a DS task at the highest priority level with utilization U_0.

Deferrable Server Bounds: We now turn to deriving least upper scheduling bounds for task sets which have a DS task as the highest priority task and n standard periodic tasks. In Section 3 we showed that the worst case phasing allows the DS task to capture the first $2C_0$ units of execution time, and $3C_0$ time units during $[0, C_0 + T_0]$. Consequently, a DS task with utilization U_0 can consume more processing time than an ordinary periodic task with utilization U_0 over the same interval. Consequently the least upper schedulability bound when a DS is included will be smaller than that derived in the previous section for a Polling Server.

The formal development of the DS case is more complex than the PS case and is included as Appendix B. We only summarize the bounds in this section.

The development of least upper bounds for the DS derived in Appendix B requires the analysis of three different cases depending upon whether: 1) T_1 and T_n are both smaller than $T_0 + C_0$, 2) T_1 and T_n are both larger than $T_0 + 2C_0$, or 3) T_1 is less than $T_0 + C_0$, but T_n is larger than $T_0 + 2C_0$.

We summarize the least upper bound on total schedulable utilization for Case 1 as a function of $R_1 = T_1/T_0$ and $U_0 = C_0/T_0$, $0 \leq U_0 \leq 1/2$.

If $T_n < T_0 + C_0$ (which entails $R_1 \leq 1 + U_0$) then we have (13) found at the bottom of the page.

The bounds corresponding to $T_0 + 2C_0 \leq T_n$ for Cases 2 and 3 may be summarized as (14) found at the bottom of the page. If we choose R_1 to minimize the above expressions, we find the least upper bounds on total schedulable utilization for the case of a high priority DS task and an arbitrary number of periodic tasks to be given by

$$\mathrm{DS}_{\mathrm{tot},\infty}(U_0) = \begin{cases} U_0 + \ln\left(\dfrac{1+U_0}{1+2U_0}(2-U_0)\right) & 0 \leq U_0 \leq \frac{1}{3} \\ U_0 + \ln(2(1-U_0)) & \frac{1}{3} \leq U_0 \leq \frac{1}{2}. \end{cases} \tag{15}$$

Note that the least upper bound is a combination of Case 1 and Case 3 with Case 3 providing the bound for $0 \leq U_0 \leq \frac{1}{3}$ and Case 1 providing the bound for $\frac{1}{3} \leq U_0 \leq \frac{1}{2}$. The Case 2 bound is greater than Case 3 over the entire range of $0 \leq U_0 \leq \frac{1}{2}$ and thus it does not contribute to the composite least upper scheduling bound given in (15).

The least upper scheduling bound given by (15) is plotted in Fig. 3 where it is compared with the least upper scheduling bounds for the high priority PS task and the limiting Liu and Layland bound of ln 2. The difference between the polling bound and the DS bound is 0 for $U_0 = 0$, 0.049 for $U_0 = 0.1$, 0.077 for $U_0 = 0.2$, 0.093 for $U_0 = 0.25$, 0.1073 for $U_0 = 0.3$ and 0.118 for $U_0 = 1/3$.

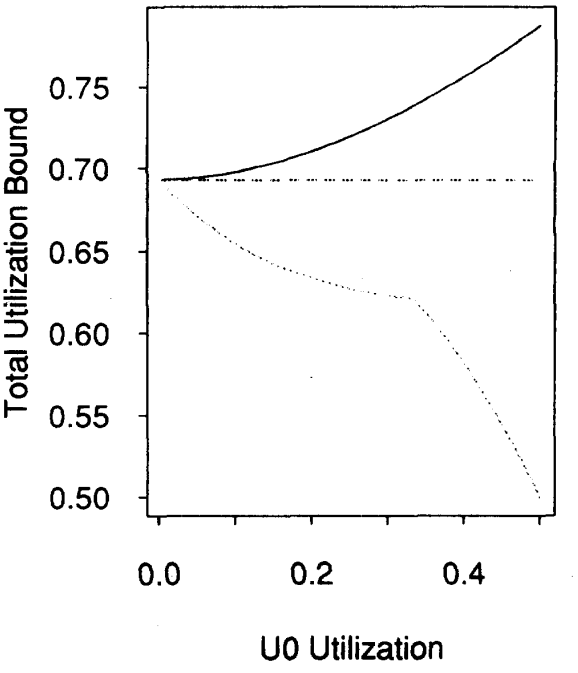


Fig. 3. Composite DS least upper scheduling bounds. Solid: polling; dotted: DS; dashed: L&L.

C. Designing a Deferrable Server

In this section, we address the question of designing a Deferrable Server for a real-time system. This question has two distinct facets: 1) determining whether to use the DS approach at all and 2) selecting the period and capacity for the DS task. The goal of any aperiodic service algorithm is to create a high priority resource for use by aperiodic tasks which will make it appear to those aperiodic tasks as if they had exclusive use of the full processing resource. To achieve such ideal behavior, the aperiodic tasks must be serviced immediately upon arrival, and their processing can only be delayed by the queuing effects of other aperiodic tasks, not by interference from the hard deadline periodic tasks. The DS task offers the promise of such transparent aperiodic service; however, only if two conditions are met: 1) the DS task must run at the highest priority level (or else it will be subject to interruptions from the periodic tasks) and 2) the capacity of the DS in each period must be sufficiently large to service a busy period of aperiodic tasks arriving during a single DS period. The first condition is simple to achieve. We need only select $T_0 \leq T_1$ to ensure that the DS task is accorded the highest rate monotonic priority.

$$\mathrm{DS}_{\mathrm{tot},\infty}(U_0, R_1) = \begin{cases} U_0 + \ln\left(\dfrac{1+U_0}{R_1}\right) + \dfrac{2R_1 - 1 - 3U_0}{1+U_0} & 0 \leq U_0 \leq \dfrac{2R_1-1}{3} \\ U_0 + \ln\left[2\left(1 - \dfrac{U_0}{R_1}\right)\right] & \dfrac{2R_1-1}{3} \leq U_0 \leq \dfrac{1}{2} \end{cases} \tag{13}$$

$$\mathrm{DS}_{\mathrm{tot},\infty}(U_0, R_1) = \begin{cases} U_0 + \ln\left(\dfrac{1+U_0}{1+2U_0}\left(2 - \dfrac{U_0}{R_1}\right)\right) & 1 \leq R_1 \leq 1+U_0 \\ U_0 + \ln\left(\dfrac{2+U_0}{1+2U_0}\right) & 1+U_0 \leq R_1 \leq 1+2U_0 \\ U_0 + \ln\left(\dfrac{2+U_0}{R_1}\right) + \dfrac{2[R_1 - (1+2U_0)]}{2+U_0} & 1+2U_0 \leq R_1 \leq 2 \end{cases} \tag{14}$$

Creating a large enough capacity for the DS task to service the aperiodic requests on demand is a more complicated issue, and whether or not it is possible will depend upon the particular task set in question.

The second condition for transparent aperiodic server operation leads one to maximize the DS task capacity subject to the periodic tasks remaining schedulable. Section V will show that the DS algorithm provides essentially ideal performance as long as the capacity of the DS task is sufficient to service aperiodic arrivals. Thus, one should use the necessary and sufficient schedulability conditions developed in Section III to maximize the size of the DS which can then provide transparent service at higher aperiodic loading levels. However, computing the maximum sized DS task consistent with meeting periodic task deadline requirements is significantly more complex for the necessary and sufficient conditions than for the least upper bound. The additional complexity associated with analyzing the necessary and sufficient conditions for the DS on-line may be prohibitive in some highly dynamic systems.

Generally, for every choice of DS period T_0 with $T_0 \leq T_1$, there is a maximum C_0 given by the necessary and sufficient schedulability test. Clearly, once T_0 has been selected, one would want to use the largest possible value for C_0. Generally, as T_0 increases, C_0 increases as well and this usually results in a larger utilization for the DS task. However, as the DS task period increases relative to the periodic tasks, anomalies can occur where a shorter period DS task could have a larger U_0, even a larger C_0 than a DS task with a slightly longer period. We next show that large values of T_0 are also desirable.

To show that large values of T_0 are desirable, suppose we compare two possible DS tasks, DS_1 and DS_2, having equal utilizations but different periods, $T_{01} < T_{02}$. There are two fundamental reasons why DS_2 is better than DS_1. First, since the two tasks have equal utilizations, $C_{01} < C_{02}$, the larger capacity DS task makes it possible to service more aperiodic tasks and/or longer aperiodic tasks without interruption. Consequently, large values of DS capacity increase the probability of aperiodic arrivals being serviced with no periodic interference, thus minimizing their response time. Second, the longer DS task period results in less wasted high priority aperiodic service capacity. To see this, consider the following example. Suppose that $C_{01} = 1$ and $T_{01} = 10$, while $C_{02} = 2$ and $T_{02} = 20$. DS_2 provides 2 units of service capacity any time during [0, 20]. Any unused capacity is lost at time 20. DS_1 also provides 2 units of service capacity during [0, 20]; however, 1 unit of capacity will be lost at time 10 if it is not used during [0, 10]. Moreover, DS_1 can only service tasks whose service requirement is no greater than 1 to completion, whereas DS_2 can service tasks whose service requirement is no greater than 2 to completion. Generally, the longer period DS task can retain high priority aperiodic service capacity over longer periods. This is the fundamental advantage that the DS task holds over the PS approach. This capability allows the DS task to better match its service to variations in aperiodic arrival pattern. Thus this advantage should be maximized.

In summary, if the largest DS task utilization that can be attained when $T_0 = T_1$ is as large as the maximum attainable DS task utilization for shorter periods, then one should select $T_0 = T_1$ and these maximum attainable utilizations should be determined using the exact schedulability equations. If, however, the attainable utilizations is significantly increased by choosing a smaller value of the DS task period, then the aperiodic performances that can be achieved with these two different choices must be directly compared. The smaller period DS task should be used only if its capacity is large enough to provide adequate continuous aperiodic service. This consideration is part of a more general issue of whether the DS should be used at all. We next turn to that question.

Once the choice of DS task has been made, one must ask whether it is appropriate to use the DS approach compared with other possible methods of providing high priority aperiodic task service. We first note that application studies [12], [10] clearly demonstrate that large aperiodic response time improvements are possible through the use of a suitable DS task. However, it should be noted that in both [12] and [10] the mean aperiodic service times were small relative to the DS task capacity, C_0. For such cases, as long as the DS task is not overloaded, most aperiodic requests will be serviced at the highest priority level which results in large reductions in aperiodic response times. More generally, one crucial comparison that must be made is the expected aperiodic busy period length versus C_0. If aperiodic arrivals are assumed to form a Poisson process and have average service requirement $E(S)$, then the mean busy period length, assuming aperiodics are not interrupted by periodics, is given by $E(B) = E(S)/(1 - \rho)$, where ρ is the traffic intensity of the aperiodic stream taken by itself. For the DS to be highly effective, one must have $E(B) < C_0$. If this inequality does not hold, then a significant fraction of the aperiodic arrivals will be serviced partially or completely at the background priority level. This reduces the effectiveness of the DS task. A Polling Server would exhibit a similar lack of effectiveness under similar conditions; however, the Polling Server has a larger capacity than the DS task. For long aperiodic requests, the effect of redistributing the aperiodic service opportunities is averaged out and the aperiodic response times converge towards background. Therefore, the DS task is not appropriate for servicing long aperiodic requests such as file transfers.

We now address the question of aperiodic response times. Given a high priority DS task with utilization U_0, determining whether aperiodic response time requirements will be met is more difficult than designing a schedulable DS task. First, consider alert class, guaranteed service. Alert class guarantees are provided assuming a minimum interarrival time for alert class tasks. Dedicated processing capacity or bandwidth is then reserved to service each alert task essentially as if it were a periodic task with a period equal to the smaller of its minimum interarrival time and its deadline. The guaranteed, alert class service provided by the DS task is very expensive from a resource allocation standpoint since the minimum interarrival time of alerts is generally much less than the average arrival time. Further, running the DS tasks at the highest priority within the rate monotonic framework requires that the DS task period to be equal to or less than the shortest period within the underlying periodic task set.

Alert class guaranteed service and soft deadline aperiodic service can be provided concurrently by using priority discrimination within the DS task, that is by reserving the required DS capacity, $C_{0,\text{alerts}}$ to be used exclusively by alert class tasks. Alternatively, one could construct two DS tasks with the same period that run at the top two priority levels in the system. These two DS tasks would then divide the available DS capacity, C_0. In either case, the highest priority must be used to service the alerts with the second highest priority used for soft deadline aperiodic service.

Generalized, analytic closed form solutions for determining whether arbitrary mixes of aperiodic soft deadline tasks will meet their desired response time requirements currently do not exist [11]. However, for the special case of small aperiodic service times (aperiodic service times much less than the DS capacity C_{DS}) and when the loading of the DS task by aperiodic arrivals is less than 70%, essentially all aperiodic arrivals are serviced at the highest priority, and an $M/M/1$ queuing analysis in which the periodic load is ignored accurately predicts soft deadline response times as will be shown in Section V. For higher DS loadings, the DS capacity is sometimes eshausted which results in some aperiodic arrivals being serviced at background priority and a queue of aperiodic tasks when the DS capacity becomes available. The simple $M/M/1$ queuing model which ignores the peiodic tasks breaks down at this point. Attempts to develop more sophisticated queuing models which accurately predict aperiodic response times for the highly loaded DS case have been unsuccessful [11].

V. Application Studies Summary

In this section we summarize the results of simulation studies which compare the response time performance of the DS algorithms against traditional Background and Polling techniques. As an additional figure of merit in evaluating the response time performance of the algorithms, we have included a lower bound on aperiodic response times. This bound is derived by assuming there is no periodic load, so all aperiodic tasks are served at high priority. The mean aperiodic response time can be calculated for this bound using an $M/M/1$ queuing analysis. The $M/M/1$ case with no periodic load was one of the cases used to validate the simulator. At the end of this section, we summarize results from other DS applications in processor scheduling and in Local Area Network Media Access (MAC) scheduling.

In each experiment, we evaluate the Polling and the DS algorithms using two distinct utilizations; one derived using necessary and sufficient scheduling conditions given by (3) and (4), and the other (smaller capacity) server utilizations derived from the least upper bounds given by (15). Each of the following nine experiments use 10 randomly selected periodic task sets with the minimum periods restricted to be greater than 55. The maximum periods for the task sets ranged from 210 to 2,310 units. The relative phasing of task periods within each task set was chosen at random. The nine experiments correspond to periodic task set loadings of 40%, 60% and 80% and aperiodic mean service times which are 1%, 2% and 5% of

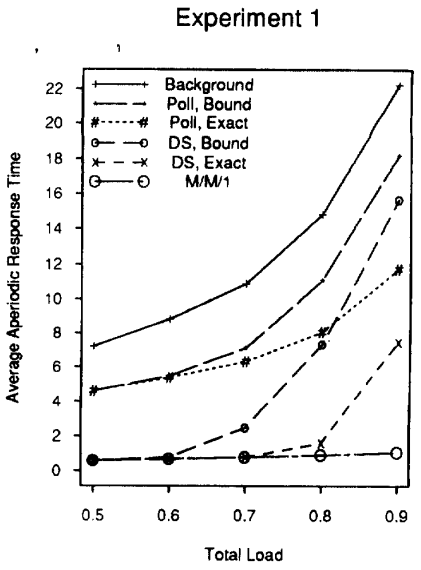


Fig. 4. Periodic load = 40%, Aperiodic mean service time = $1/\mu = 0.55$.

the server's period (55). The aperiodic task arrival times were modeled using a Poisson arrival process, and the aperiodic task service times were modeled using an exponential service time distribution. The set of nine experiments allows us to evaluate the sensitivity of the DS algorithm to both periodic loading and mean aperiodic service times.

Fig. 4 summarizes the results of Experiment 1. In this and all other figures "exact" refers to a DS or PS determined using necessary and sufficient schedulability conditions while "bound" refers to a DS or PS determined by least upper bounds. For this experiment, the periodic load was held constant at 40%, while the aperiodic load was increased from 10% to 50% (resulting in a total load ranging from 50% to 90%). Using the least upper bounds for the Polling Server and Deferrable Server resulted in server capacities of 14.75 and 12.62 units respectively during each 55 unit server period. The exact schedulability conditions resulted in server capacities that varied from 27.41 to 32.94 units for Polling, and 20.38 to 23.71 units for the Deferrable Server for the 10 randomly selected periodic task sets used in this experiment. In the following discussions, it is useful to consider the traffic intensities on the servers when evaluating their relative performance. We define the server traffic intensity as TI = U_{ap}/U_s where U_{ap} is the aperiodic load and $U_s = C_s/T_s$ = *Utilization of the server*. When TI is less than 70% and $E(S) < C_0$, then most aperiodic requests are serviced by the DS task without any interference from the periodic tasks. As TI increases beyond this level, an increasing fraction of the aperiodics will be serviced as background tasks which will substantially degrade aperiodic responsiveness.

Both the Deferrable Servers provide nearly optimal aperiodic response time performance up to a total of 60%. These response times are nearly optimal in the sense that they are very close to the performance that would be obtained in the absence of a periodic load component. Note that the

aperiodic tasks are serviced at the full processor speed, not at a degraded rate proportional to the server's utilization. In this range, for small aperiodic task sizes, the aperiodic average response times can be accurately predicted by simple $M/M/1$ queuing models that ignore the periodic load component. Over this region the DS algorithm enjoys more than a factor of 10 response time advantage over Background, and more than a factor of 7 response time advantage over Polling.

As the total load increases over 60%, the servers' performance diverge from the ideal with the smaller utilization "bound" servers degrading more quickly than the larger "exact" servers. At the 70% total load point, the traffic intensity on the "bound" Polling and DS servers are 112% and 130% respectively. Unlike conventional queuing systems where traffic intensities in excess of 100% result in an infinite expected queue length, the Polling and DS servers assign Background priority to arrivals when their respective capacities are exhausted. These servers thus provide either highly responsive aperiodic service when there is sufficient capacity, or Background service when their capacities are exhausted. The DS "exact" server, with a traffic intensity of about 75%, still enjoys performance nearly equivalent to the $M/M/1$ performance with essentially all aperiodic arrivals being serviced at high priority. The higher capacity "exact" DS server provides more than a factor of three performance advantage over the "bound" DS server, and advantage factors of more than 7, 8 and 13 over Polling "exact", Polling "bound", and Background respectively at the 60% total load point.

When the total loading is increased to 80% the DS "exact" performance has diverged slightly from the $M/M/1$ lower bound. At this point, the traffic intensity on the DS "exact" server is 100%, and a significant fraction of the aperiodic arrivals are assigned Background priority. Near the 80% total load point the DS "bound" and PS "exact" performance measures intersect. At this loading, the traffic intensity on the DS "bound" is 174% which, on average, results in 42% of the aperiodic arrivals being serviced at Background priority. The traffic intensity of the PS "exact" is only 73%. At this point, the relative server size advantage of the Polling "exact" (30 units to 12.62 units) outweighs the Deferrable Server's capability to defer aperiodic service until needed. The larger capacity DS "exact" still maintains a substantial performance advantage over the other servers with advantage factors of more than 4, 5, 6, and 9 over DS "bound", PS "exact," PS "bound," and Background.

At 90% total load, the DS "exact" is saturated with a traffic intensity of 125%, and its performance sharply degrades. The other servers similarly continue to degrade with the DS "bound" performance converging towards PS "bound." The preceding experiment demonstrated the ability of the DS algorithm to convert excess periodic task slack time into highly responsive aperiodic service. For this case, the DS "exact" provided response time service nearly identical to the expected response times in the absence of periodic loading. In effect, the DS algorithm provides nearly optimal service up to the loading at which it becomes nearly saturated, at which point its response time starts to fall off sharply as a significant portion of the aperiodic arrivals must be serviced at Background priority.

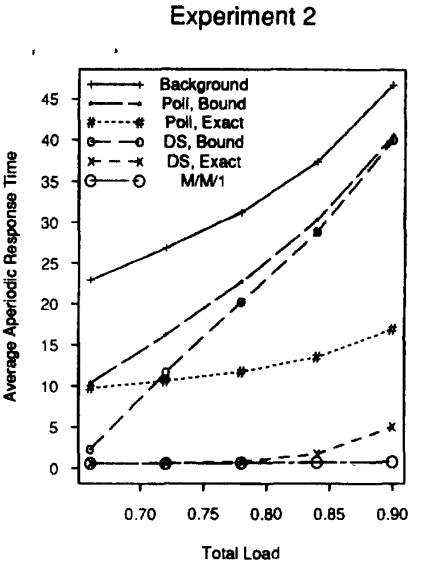


Fig. 5. Periodic load = 60%, Aperiodic mean service time = $1/\mu = 0.55$.

For Experiment 2, illustrated in Fig. 5, we increased the periodic load component to 60%. The aperiodic load was then increased from 5% to 30% for a total loading range from 65% to 90%. Using the least upper bounds for the Polling Server and Deferrable Server results in server capacities of 5.37 and 3.61 units respectively during each 55 unit server period. The exact server capacities varied from 21.99 to 15.31 units for Polling, and 19.92 to 11.16 units for the Deferrable Server across the 10 randomly selected periodic task sets used in this experiment. Again, the aperiodic response time in the absence of the periodic load component is plotted for comparison. Once again, we note that the larger capacity DS "exact" provides nearly optimal service across most of the range until it becomes saturated.

At 65% total load (5% aperiodic load), the PS and DS "bound" servers' performance are already diverging from their corresponding "exact" servers. This is because of the very small capacities for the "bound" servers with their resultant traffic intensities of 101% and 138% respectively. As we increase the aperiodic load above 5% the "bound" servers' performance quickly degrades towards Background with a relatively low percentage of aperiodic arrivals being provided high priority service. The bulk of these arrivals are scheduled at Background priority which results in their average performance converging toward the average Background performance. In contrast, the "exact" servers have sufficient capacity to service the majority of aperiodic arrivals at high priority thus creating a widening performance advantage over Background.

The DS "exact" provides the best response time performance maintaining essentially the $M/M/1$ ideal performance out to about 78% total loading. At 65% total loading the

performance advantage factors are more than 3, 16, 17, and 39 respectively compared to DS "bound", PS "exact," PS "bound", and Background respectively. The DS "bound" performance intersects the larger PS "exact" performance at about 72% total load. As the aperiodic load is increased to 13% (78% total load), the DS "exact" performance advantage further widens to more than 15, 25, 29, and 40 over PS "exact", DS "bound", PS "bound", and Background respectively. At a 13% aperiodic load the traffic intensities on the PS and DS "exact" servers average 53% and 64% respectively thus accounting for their ideal performance up to this loading level. The PS "bound" performance parallels the DS "exact" performance across the range with each degrading as the servers start to saturate in the mid 80% total loading range. At 90% total load the DS "exact" performance advantage factors are in excess of 3, 7, 8 and 9 for PS "exact" DS "bound," PS "bound" and Background respectively.

For Experiment 3, illustrated in Fig. 6, we further increased the periodic load component to 80%. The aperiodic load was then increased from 2% to 10% for a total loading range from 82% to 90%. At these high periodic utilization levels, it is not possible to use the "bound" servers, and it is necessary to use explicit task information to determine the "exact" servers for DS and Polling. The "exact" server capacities varied from 10.54 to 1.82 units for Polling, and 10.73 to 1.39 units for the Deferrable Server across the 10 randomly selected periodic task sets used in this experiment. The DS "exact" provides the best performance across the range diverging relatively slowly from $M/M/1$ ideal performance as the DS server becomes saturated. The DS "exact" performance is from 2 to 16 times better than the PS "exact" performance, and 9 to 70 times better than Background performance.

It is useful to jointly compare the results of the last set of three experiments at 40%, 60% and 80% periodic loading. Adjusting to the differing scales needed to accommodate the widening relative ranges, the DS and PS "exact" performances were nearly invariant to the increasing periodic load. In essence these "exact" servers were of sufficient capacity to service nearly all aperiodic arrivals at high priority. In contrast, the lower capacity "bound" servers' performance degraded quickly with increasing periodic load. This indicates that there are significant performance advantages to using the largest possible server as determined by necessary and sufficient schedulability conditions. Whenever possible, one should use the larger "exact" servers.

The above set of three experiments evaluated the relative performance of the DS algorithm for short service times and increasing periodic load. We also evaluated the algorithm for increasing aperiodic mean service time and found the DS performance to degrade with increasing aperiodic service times. Fig. 7 illustrates the results for the composite set of nine experiments. Each row reflects increasing periodic load components of 40%, 60% and 80%. Each column reflects increasing aperiodic mean service times with the mean service time doubling in each subsequent row. Experiments 1 through 3 reflects the results for the relatively short aperiodic service times already discussed above. Experiments 1, 4, and 7 summarize the results of increasing aperiodic mean service

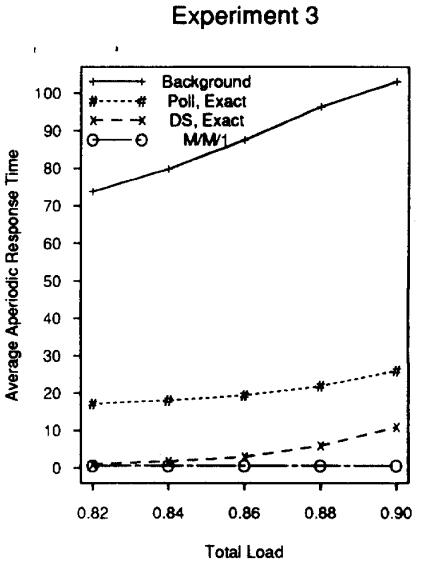


Fig. 6. Periodic load = 80%, Aperiodic mean response time = $1/\mu = 0.55$.

time with the periodic load component at 40%. As we increase the mean service time, the servers' response time performance degrade more quickly from the lower bound $M/M/1$ performance. Note that the servers' traffic intensities are constant at corresponding points among the three graphs. Thus the servers' performance are sensitive not only to loading, but also to aperiodic job size.

As we examine the graphs representing 80% periodic load, we note that with increasing aperiodic mean service times the performance of the "exact" servers move further away from $M/M/1$ performance as the service time is increased. However, in all cases, DS "exact" provides the best performance over the broadest range with significant performance advantages in all cases. In contrast, DS "bound" performance (columns 1 and 2) starts out nearly optimal, but degrades toward Polling as the load is increased with the PS "exact" providing better performance at high loading. At this point the relative size advantage of the PS "exact' outweighs the capability of the DS server to defer service until needed. For each of the experiments, the DS "exact" provided the best performance. In all cases the "exact" servers' relative performance improved over Background as the periodic loading was increased. In effect, the servers have the ability to break up large periodic task busy periods while still meeting all periodic task deadlines.

The above experiments demonstrate the DS algorithm's ability to reduce aperiodic response times in hard real-time environments. In all cases the algorithm maintained guaranteed periodic response times while minimizing aperiodic response times by converting excess periodic slack time into high priority aperiodic service. Earlier simulation studies for processor scheduling demonstrated that the DS algorithm could significantly reduce aperiodic response times while maintaining guaranteed periodic task deadlines. Response time improvements of 90% were demonstrated (12).

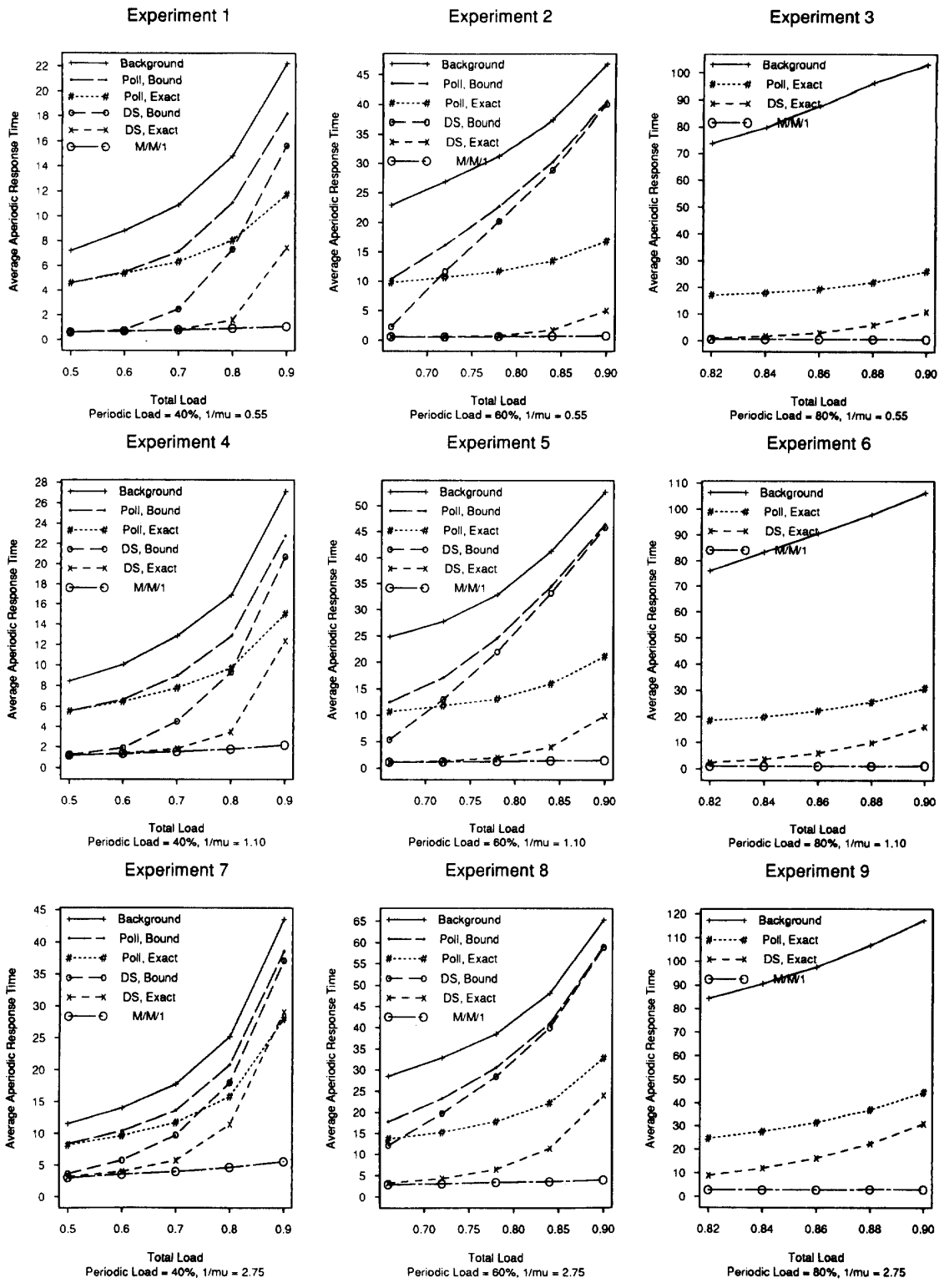


Fig. 7. Composite experiments.

The above experiments demonstrated the capability of the DS algorithm to greatly reduce soft deadline aperiodic response times in hard deadline, real-time systems. The DS algorithm can also be used to provide guaranteed, highly responsive (alert class) aperiodic service. This latter capability was demonstrated in application studies in the Local Area Network (LAN) media access (MAC) scheduling for both IEEE 802.5 Token Rings (10) and SAE-9B High Speed Ring Bus (HSRB) (13, 12). In both application studies, media access scheduling models supporting global, prioritized contention resolution were first developed. The DS algorithm was then applied. In both of the LAN studies the DS algorithm provided guaranteed alert class aperiodic deadlines and greatly enhanced aperiodic response times while still maintaining guaranteed periodic deadlines. The response time improvement for the soft deadline aperiodic tasks was nearly two orders of magnitude in both cases.

VI. Conclusion

This paper has presented the theoretical foundations for the Deferrable Server algorithm which provides a solution to the problem of jointly scheduling hard deadline periodic tasks and hard and soft deadline aperiodic tasks. To provide a fair comparison, both necessary and sufficient conditions and least upper bounds were developed for the Deferrable Server algorithm and the more conventional polling technique of scheduling aperiodic service in hard-time environments. Taking advantage of the fact that there is typically no advantage for the system for periodic tasks completing early, the DS algorithm converts the excess periodic task slack time into highly responsive aperiodic class performance. The algorithm has been used to introduce highly responsive, guaranteed alert task aperiodic service while still maintaining periodic task guarantees as well as providing response time improvements of an order of magnitude for soft deadline aperiodic tasks. The algorithm has been shown to provide nearly optimal aperiodic response time performance for relatively short aperiodic mean service times up to very high server traffic intensities. As the mean service times were increased, the DS response time performance diverged more quickly from the optimal non-interfering case. Other application simulation studies in both processor and LAN media access scheduling have shown similar results and sensitivities.

Appendix A
Polling Server Least Upper Bounds

Section IV developed least upper schedulability bounds for the case of two tasks where one of the tasks was a high priority Polling Server, PS, task. The schedulability bounds for the general case of n periodic tasks were then summarized in (4). The supporting proofs for the PS general case are provided in this appendix.

The analysis essentially follows that of Liu and Layland. One considers *full-utilization tasks sets* in which any one of the tasks has a fixed utilization of U. These are task sets which are schedulable, but if the computation requirement of any of the tasks were to be increased, a deadline would be missed. We seek the full-utilization task set with minimum utilization. The utilization of that task set will be $\mathrm{PS}_n(U)$. We refer to such a task set as a *worst case task set.*

Assume $T_0 \le T_1 \le \cdots \le T_n$. We first show that we can restrict attention to task sets that satisfy $T_n/T_0 < 2$. Suppose instead $T_n/T_0 \le 2$ and one of the tasks has utilization fixed at U. Then if the utilization of neither τ_0 nor τ_n has been fixed, a full-utilization task set with less utilization can be constructed by modifying τ_0 and τ_n as follows: assume $T_n = kT_0 + r, k \ge 2, 0 \le r < T_0$, and let $T_0^\star = kT_0, C_0^\star = C_0, T_n^\star = T_n$ and $C_n^\star = C_n + (k-1)C_0$. The new task set has less total utilization and is either full-utilization or a deadline is missed and the utilization of the other periodic tasks must be reduced to achieve schedulability. If τ_0 has utilization U_0, then one can modify τ_0 by letting $T_0^\star = kT_0$, and $C_0^\star = kC_0$ and apply the preceding argument if $T_n/T_1 \ge 2$. The result is a full-utilization task set with no greater utilization. If τ_n has utilization U, then one can define $T_j^\star = \lfloor \frac{T_n}{T_j} \rfloor T_j$ and $C_j^\star = \lfloor \frac{T_n}{T_j} \rfloor C_j, 0 \le j \le n-1$. The result is a task set satisfying $T_n/T_0' < 2$ which is either full utilization or whose utilization must be further reduced so that deadlines are not missed. Thus we can restrict attention to the case $T_n/T_0 \le 2$.

The second step is to characterize the worst case task set assuming $T_n/T_0 < 2$ with one task having a utilization U. Using the argument of Liu and Layland we find $C_i = T_{i+1} - T_i$, $0 \le i \le n-1$ and $C_n = 2T_0 - T_n$ to be the choice of C_i resulting in the minimum utilization. The resulting total utilization is given by

$$\sum_{i=0}^{n-1} \frac{T_{i+1} - T_i}{T_i} + \frac{2T_0 - T_n}{T_n}.$$

Defining $R_i = T_i/T_{i-1}$, $1 \le i \le n$, this expression becomes

$$\sum_{i=1}^{n} (R_i - 1) + 2/(R_1 \cdots R_n) - 1 = \sum_{i=1}^{n} R_i + 2 \Big/ \prod_{i=1}^{n} R_i - (n+1). \tag{16}$$

Here, $R_i \ge 1, \prod_{i=1}^n R_i < 2$ and one of the tasks has utilization U. The latter condition implies $R_i = 1+U$ for some $1 \le i \le n$ or $2/\prod_{i=1}^n R_i = 1 + U$.

Some of the subsequent analysis is based on the following lemma:

Lemma A.1: Let $x > 0$, $y > 1$ and $H(\mathbf{R}) = \sum_{i=1}^n R_i + x/\prod_{i=1}^n R_i - (n+1)$, where $\mathbf{R} = (R_1, \cdots, R_n)$. Let $\mathbf{S}_y = \{\mathbf{R} | R_i \ge 1, \prod_{i=1}^n R_i \le y\}$. Then

$$\min_{\mathbf{R} \in \mathbf{S}_y} H(\mathbf{R}) = \begin{cases} (n+1)(x^{1/(n+1)} - 1), & \text{if } x^{n/(n+1)} \le y \\ n(y^{1/n} - 1) + (x-y)/y, & \text{if } y < x^{n/(n+1)} \end{cases} \tag{17}$$

$$\lim_{n \to \infty} \min_{\mathbf{R} \in \mathbf{S}_y} H(\mathbf{R}) = \begin{cases} \ln x, & x \le y \\ \ln y + (x-y)/y, & x > y. \end{cases} \tag{18}$$

Proof: Consider any $\mathbf{R}\epsilon\mathbf{S}_y$ with $R_i < R_j$ for some $i \neq j$. Define $\mathbf{R}^\star$ by letting $R_i^\star = R_j^\star = (R_iR_j)^{1/2}$ and $R_k^\star = R_k$, $k \neq i,j$. It follows that $\mathbf{R}^\star\epsilon\mathbf{S}_y$ and $H(\mathbf{R}) - H(\mathbf{R}^\star) = R_i + R_j - R_i^\star - R_j^\star = R_i + R_j - 2(R_iR_j)^{1/2} = (R_i^{1/2} - R_j^{1/2})^2 > 0$. Consequently, the minimizing $\mathbf{R}$ must be of the form $\mathbf{R} = R(1,1,\cdots,1)$. This reduces consideration to the function $H(R) = nR + x/R^n - (n+1)$ subject to $1 \leq R \leq y^{1/n}$. Clearly

$$H'(R) = n(1 - x/R^{n+1}) = \begin{cases} < 0 & \text{if } R\epsilon(0, x^{1/(n+1)}), \\ = 0 & \text{if } R = x^{1/(n+1)} \\ > 0 & \text{if } R\epsilon(x^{1/(n+1)}, \infty). \end{cases} \quad (19)$$

Therefore the minimum of $H(R)$ subject to $R \leq y^{1/n}$ occurs at $R = \min(x^{1/(n+1)}, y^{1/n})$. Substituting R into $H(R)$ gives Eq. 17. Equation (18) follows from the fact that $\lim_{n\to\infty} n(x^{1/n} - 1) = \ln x$ for $x > 0$. □

A straightforward application of Lemma A.1 applied to (16) with either $R_i = 1+U$ for any particular $1 \leq i \leq n$ or a simple minimization argument $2/\prod_{i=1}^n R_i = 1 + U$ yields the least upper bound on the total utilization where any one task has a fixed utilization U. We replace U by U_0 for later comparisons with the DS algorithm and get

$$\mathrm{PS}_{\mathrm{tot},n}(U_0) = n\left(\left(\frac{2}{1+U_0}\right)^{1/n} - 1\right) + U_0. \quad (20)$$

Equation (20) has two terms, the first gives the utilization of the periodic tasks, while the second corresponds to the utilization of the Polling Server. Letting $n \to \infty$ one can find the limiting value for the least upper bound,

$$\mathrm{PS}_{\mathrm{tot},\infty}(U_0) = \ell n\left(\frac{2}{1+U_0}\right) + U_0. \quad (21)$$

Equation (21) is a generalization of the Liu and Layland bound of $\ell n2$ in which one of the tasks has utilization fixed to be U_0. A graph of this bound is given in Fig. 8. One can see that there is very little change until U_0 becomes very large. For example, when $U_0 = 0.25$, $\mathrm{PS}_{t,\infty}(U_0) = 0.720$, only slightly larger than $\ell n = 0.693$, the usual Liu and Layland bound.

Equation (20) shows that if one has a polling task at any priority level with utilization U_0, then all periodic task sets consisting of n standard periodic tasks (in addition to the polling task) with total utilization no greater than $n\left(\left(\frac{2}{1+U_0}\right)^{1/n} - 1\right)$ will be schedulable. We now wish to determine a similar expression when the polling task is replaced by a DS task at the highest priority level with utilization U_0.

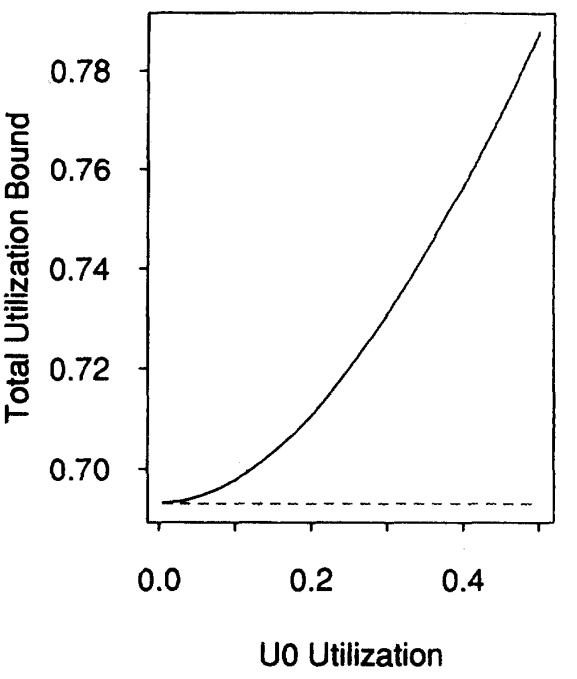


Fig. 8. Polling server least upper scheduling bound. Solid: polling; dashed: L&L.

Appendix B
Deferrable Server Least Upper Bounds

Section IV developed least upper schedulability bounds for the case of two tasks where one of the tasks was a high priority Deferrable Server, DS, task. The schedulability bounds for the general case of n periodic tasks were then summarized. The supporting proofs for the DS general case are provided in this appendix.

We consider a task set with a high priority DS task, τ_0, having utilization U_0 and n periodic tasks $\tau_1,\cdots,\tau_n$ with $T_0 \leq T_1 \leq \cdots \leq T_n$. The determination of a worst case task set is facilitated by examining restrictions to the period ratios. For example, Appendix A showed that $T_n/T_0 < 2$ is a necessary condition for a worst case task set. When a DS task, τ_0 is included at the highest priority, the worst case period ratios change. There are two necessary condition for a task set to be a worst case task set: $T_n/T_1 < 2$ and $T_n/T_0 < 2 + U_0$. These conditions can be derived as follows.

Suppose $\tau_0,\tau_1,\cdots,\tau_n$ is a task set that fully utilizes the processor, and suppose $T_n/T_1 \geq 2$, that is $T_n = kT_1 + r$, $0 \leq r < T_1$ with $k \geq 2$. Assuming the worst case phasing, we modify τ_1 and τ_n to τ_1^* and τ_n^* as follows. Let $C_1^* = C_1, T_1^* = kT_1, C_n^* = C_n + (k-1)C_1$ and $T_n^* = T_n$. The modified task set fully utilizes the processor, but the utilization of the periodic task set is reduced by $(k-1)C_1(1/T_n - 1/kT_1) > 0$. Consequently $T_n/T_1 < 2$ is a necessary condition for the worst case task set.

We next show that $T_n/T_0 < (2 + U_0)$ is a necessary condition for a worst case task set. Suppose $C_0 + kT_0 \leq T_n < 2C_0 + kT_0$ for some $k \geq 1$. In the worst case, the DS task, τ_0, would run during $[C_0 + kT_0, 2C_0 + kT_0]$, thus T_n could be increased to $2C_0 + kT_0$, the task set would still be full utilization and its total utilization would be reduced. Consequently, attention can be restricted to the case $2C_0 + kT_0 \leq T_n < C_0 + (k+1)T_0$ for $k \geq 1$. Suppose $2C_0 + kT_0 < T_n < C_0 + (k+1)T_0$ for some $k \geq 2$. We can

modify the DS task to $T_0' = T_n/(2U_0 + k)$ and $C_0' = U_0 T_0'$. Note that $T_0' < T_1$ because $T_n/T_1 < 2$, so the DS task retains the highest priority. The modified DS task runs the same number of times $(k + 2)$ during $[0, T_n]$, but each is longer. Consequently, the periodic utilization must be reduced to maintain schedulability. Therefore, $2C_0 + kT_0 < T_n < C_0 + (k+1)T_0$ cannot give the worst case if $k \geq 2$. It remains to consider the case $T_n = 2C_0 + kT_0, k \geq 2$. If $k \geq 3$, we modify the DS task so that $T_0' = T_0\left(\frac{2U_0+k}{2U_0+k-1}\right)$, and $C_0' = U_0 C_0$. The utilization is preserved, $T_0' < T_n/2 < T_1$ so the DS task retains the highest priority. The modified DS task fills more of $[0, T_n]$ than the original, so the periodic utilization must be reduced to retain schedulability. This leaves only one case to be considered, $T_n = 2C_0 + 2T_0$. Later in this section, we will give an explicit computation of the worst case utilization for this case and show that it is larger than that for the other cases. Once this is done, we will have shown $T_n/T_0 < 2 + U_0$ is a necessary condition for worst case periodic task set utilization.

We now turn to developing least upper bounds on the schedulable periodic task utilization as a function of U_0, the utilization of the high priority DS task. We assume $T_0 \leq T_1 \leq \cdots \leq T_n, T_n/T_0 < 2$ and $T_n/T_0 < 2 + U_0$. The analysis is carried out by considering three distinct cases:

Case 1 :$T_0 \leq T_1 \leq \cdots \leq T_n < T_0 + C_0$

Case 2 :$T_0 \leq T_0 + C_0 \leq T_1 \leq \cdots \leq T_n \leq 2T_0 + C_0$

Case 3 :$T_0 \leq T_1 \leq \cdots T_k \leq T_0 + C_0 \leq T_0 + 2C_0$
$\leq T_{k+1} \leq \cdots \leq T_n \leq 2T_0 + C_0$
for some $k, 1 \leq k \leq n-1$.

The goal is to find upper bounds for each of these three cases as a function of n, U_0 and $R_1 = T_1/T_0$. These three bounds are themselves of independent interest. Minimization over the three bounds yields the least upper bound on utilization for the periodic tasks as a function of U_0.

Case 1 Bounds: We begin with the simplest case in which for the worst case phasing, the DS task occupies only the interval $[0.2C_0]$ before T_n, the deadline of τ_n. Specifically, we assume $T_0 \leq T_1 \leq \cdots \leq T_n \leq T_0 + C_0$, the task set fully utilizes the processor, and T_0 and C_0 are fixed with $C_0/T_0 = U_0$. We seek values of $(C_i, T_i), 1 \leq i \leq n$ satisfying the above restrictions for which $\sum_{i=1}^{n} C_i/T_i$ is minimized.

The following lemma which is based on a similar lemma of Liu and Layland gives the minimizing values of $C_i, 1 \leq i \leq n$ as a function of $T_i, 0 \leq i \leq n$.

Lemma B.1: Suppose a task set satisfies $T_0 \leq T_1 \leq \cdots \leq T_n < T_0 + C_0$. Under the worst case phasing, $\sum_{i=1}^{n} C_i/T_i$ is minimized by $C_i = T_{i+1} - T_i, 1 \leq i \leq n-1$ and $C_n = 2(T_1 - C_0) - T_n$.

Proof: Suppose $C_1 = T_2 - T_1 - \epsilon \geq 0$ for some $\epsilon > 0$. Since the task set fully utilizes the processor, the time interval $[T_1 + C_1, T_2]$ must be utilized by lower priority tasks. Suppose τ_2 utilizes the ϵ units of computation. Consider the modified task set in which only τ_1 and τ_2 are modified by $C_1^\star = C_1 + \epsilon/2$ and $C_2^\star = C_2 - \epsilon$. The modified task still fully utilizes the processor but has utilization reduced by $\epsilon(-1/2T_1 + 1/T_2) > 0$. Consequently, for minimum utilization we must have $\epsilon = 0$. Continuing in a similar fashion, no lower priority task can execute during $[T_1 + C_1, T_2]$ and still have minimum utilization. Consequently, a necessary condition is $C_1 \geq T_2 - T_1$.

Next, suppose $C_1 = T_2 - T_1 + \epsilon$ for some $\epsilon > 0$. Consider the modified task set in which only τ_1 and τ_2 are modified by $C_1^\star = T_2 - T_1, C_2^\star = C_2 + \epsilon$. The modified task set fully utilizes the processor, but the utilization is reduced by $\epsilon(1/T_1 - 1/T_2) > 0$. Consequently, a necessary condition is $C_1 \leq T_2 - T_1$. It follows that $C_2 = T_2 - T_1$. By repeating this identical argument sequentially for $C_2, \cdots, C_n$, we find $C_i = T_{i+1} - T_i, 1 \leq i \leq n-1$. In order for the task set to fully utilize the processor, $C_n = 2(T_1 - C_0) - T_n$, and the lemma follows. □

From Lemma B.1, we know that the worst case periodic utilization corresponds to a task set with computation requirements given by:

$$C_i = T_{i+1} - T_i, 2 \leq i \leq n-1 \tag{22}$$

$$C_n = 2(T_2 - C_1) - T_n. \tag{23}$$

The periodic utilization for these tasks is given by

$$U_{\text{per}} = \sum_{i=1}^{n-1} \frac{(T_{i+1} - T_i)}{T_i} + \frac{[2(T_1 - C_0) - T_n]}{T_n} \tag{24}$$

$$= \sum_{i=1}^{n-1} R_{i+1} + \frac{2(R_1 - U_0)}{R_1 \cdots R_n} - n \tag{25}$$

where $R_i = T_i/T_{i-1}, 1 \leq i \leq n$.

$$U_{\text{per}} = \sum_{i=2}^{n} R_i + \frac{2(1 - \frac{U_0}{R_1})}{R_2 \cdots R_n} - n \tag{26}$$

subject to $R_1 \cdots R_n \leq 1 + U_0$ or $R_2 \cdots R_n \leq (1 + U_0)/R_1$. Using Lemma A.1, the bound for n periodic tasks as a function of R_1 and U_0 is given by

$$\text{DS}_{\text{per},n}(U_0, R_1) = \begin{cases} (n-1)\left(\left(\frac{1+U_0}{R_1}\right)^{\frac{1}{n-1}} - 1\right) & \\ \quad + \frac{2R_1 - 1 - 3U_0}{1+U_0} & \text{if } \left(2\left(1 - \frac{U_0}{R_1}\right)\right)^{\frac{n-1}{n}} > \frac{1+U_0}{R_1} \\ n\left(\left[2(1 - \frac{U_0}{R_1})\right]^{\frac{1}{n}} - 1\right) & \text{if } \left(2\left(1 - \frac{U_0}{R_1}\right)\right)^{\frac{n-1}{n}} \leq \frac{1+U_0}{R_1} \end{cases} . \tag{27}$$

Letting $n \to \infty$ we find

$$\text{DS}_{\text{per},\infty}(U_0, R_1) = \begin{cases} \ln\left(\frac{1+U_0}{R_1}\right) + \frac{2R_1 - 1 - 3U_0}{1+U_0} & \text{if } 0 \leq U_0 \leq \frac{2R_1 - 1}{3} \\ \ln\left(2(1 - \frac{U_0}{R_1})\right) & \text{if } \frac{2R_1 - 1}{3} \leq U_0 \leq 1/2 \end{cases} . \tag{28}$$

The above bounds can be further minimized by finding a minimizing value of $R_1 \in [1, 1 + U_0]$. The bounds are minimized by setting $R_1 = 1$ that is by allowing the DS period to be as large as possible while maintaining the highest priority. In this case, the asymptotic bounds on periodic task utilization become

$$\text{DS}_{\text{per},\infty}(U_0) = \begin{cases} \ln(1 + U_0) + \frac{1 - 3U_0}{1+U_0} & 0 \leq U_0 \leq \frac{1}{3} \\ \ln(2(1 - U_0)) & \frac{1}{3} \leq U_0 \leq 1/2 \end{cases} . \tag{29}$$

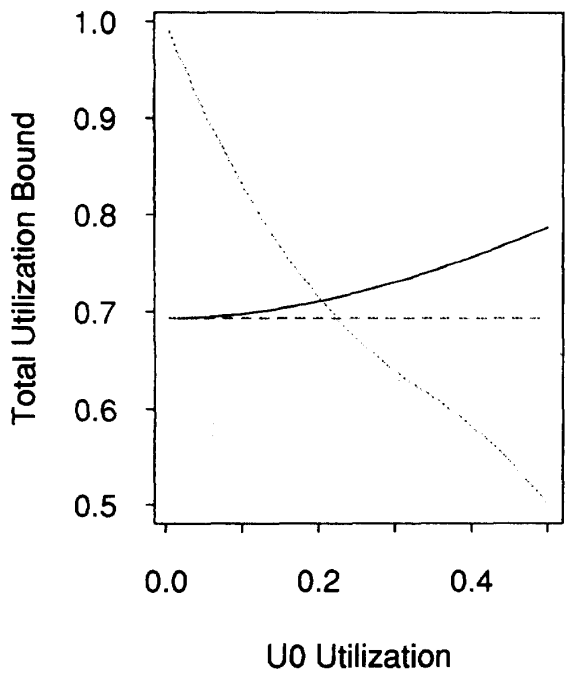


Fig. 9. Case 1 bound. Solid: polling; dotted: DS; dashed: L&L.

Adding U_0 gives the least upper bound for total schedulable utilization of

$$\mathrm{DS}_{\mathrm{tot},\infty}(U_0)=\begin{cases} U_0+\ln(1+U_0)+\frac{1-3U_0}{1+U_0} & 0\le U_0\le \frac{1}{3} \\ U_0+\ln(2(1-U_0)) & \frac{1}{3}\le U_0\le \frac{1}{2} \end{cases}. \tag{30}$$

Fig. 9 plots the Case 1 bounds along with the bounds corresponding to a PS, and the generalized Liu and Layland case. If we restrict attention to $0 \le U_0 \le 1/3$, then the worst case total utilization in Case 1 is

$$\mathrm{DS}_{\mathrm{tot},\infty}(U_0) = U_0 + \ln(1+U_0) + \frac{1-3U_0}{1+U_0}. \tag{31}$$

This is a decreasing function of U_0. The total schedulable utilization for $1/3 \le U_0 \le 1/2$ is given by

$$\mathrm{DS}_{\mathrm{tot},\infty}(U_0) = U_0 + \ln(2(1-U_0)). \tag{32}$$

This bound is .621 when $U_0 = 1/3$ and decreases as U_0 increases to $1/2$ to its smallest value, .5. Indeed, the worst case task set is given by the special situation in which $T_0 = 2C_0, T_0 = \cdots = T_n$ and $C_1 = \cdots = C_n = 0$.

In case 1, if $U_0 \le 0.2$, the schedulable utilization is at least 0.7157, and reaches 1.0 as U_0 decreases to 0. The 100% schedulability occurs because the defining condition for Case 1 is $T_n \le T_0 + C_0$, and as U_0 decreases the periodic task periods converge to a common value. In this case, the rate monotonic algorithm can schedule loads up to 100%.

Case 2 Bounds: The second case to be considered is characterized by $T_0 + C_0 \le T_1 < \cdots < T_n \le 2T_0 + C_0$, that is the deadlines of the periodic tasks lie between the third occurrence of the DS task and the start of its fourth occurrence. It should be clear that the worst case task set will satisfy $T_0 + 2C_0 \le T_1$. The proof of Lemma B.1 can be easily modified to show that the worst case computation times for this case are given by

$$C_i = T_{i+1} - T_i, \quad 1 \le i \le n-1 \tag{33}$$

$$C_n = 2T_1 - T_n - 3C_0 \tag{34}$$

The periodic utilization for this task set is given by

$$U_{\mathrm{per}} = \sum_{i=1}^{n-1} \frac{T_{i+1}-T_i}{T_i} + \frac{2T_1 - T_n - 3C_0}{T_n} \tag{35}$$

$$= \sum_{i=2}^{n} R_i + \frac{2 - 3C_0/T_1}{R_2 \cdots R_n} - n \tag{36}$$

for $R_i = T_{i+1}/T_i$ subject to $R_1 \ge 1 + 2U_0, R_1 \cdots R_n \le 2 + U_0$ or $R_2 \cdots R_n \le (2+U_0)/R_1$. Consequently,

$$U_{\mathrm{per}} = \sum_{i=2}^{n} R_i + \frac{2 - 3U_0/R_1}{R_2 \cdots R_n} - n. \tag{37}$$

This expression can be minimized using Lemma A.1. We find the minimum value to be given by (38) found at the bottom of the page. Letting $n \to \infty$ we find (38) reduces to $\mathrm{DS}_{\mathrm{per,infty}}(U_0, R_1) = \lim_{n\to\infty} \mathrm{DS}_{\mathrm{per},n}(U_0, R_1)$:

$$\mathrm{DS}_{\mathrm{per},\infty}(U_0, R_1) = \ln\left(\frac{2+U_0}{R_1}\right) + \frac{2[R_1 - (1+2U_0)]}{2+U_0} \quad \text{for } 1+2U_0 \le R_1 \le 2+U_0. \tag{39}$$

This bound is minimized by selecting R_1 as small as possible, thus we set $R_1 = 1 + 2U_0$. We find the least upper bound for the periodic component for Case 2 to be given by

$$\mathrm{DS}_{\mathrm{per},\infty}(U_0) = \ln\left(\frac{2+U_0}{1+2U_0}\right). \tag{40}$$

The total schedulable utilization is plotted in Fig. 10, and is given by

$$\mathrm{DS}_{\mathrm{per},\infty}(U_0) = U_0 + \ln[(2+U_0)/(1+2U_0)]. \tag{41}$$

The DS Case 2 bound is minimized when $U_0 = \frac{\sqrt{33}-5}{4} = 0.186$ which gives a schedulable utilization of 0.652, slightly below the Liu and Layland bound of 0.693 and the polling bound of 0.708 given by (12) for a Polling Server with capacity 0.186.

$$\mathrm{DS}_{\mathrm{per},n}(U_0, R_1) = \begin{cases} n\left[\left(2 - 3\frac{U_0}{R_1}\right)^{\frac{1}{n}} - 1\right] & \text{if } \left(2 - 3\frac{U_0}{R_1}\right)^{\frac{n-1}{n}} \le \frac{(2+U_0)}{R_1} \\ (n-1)\left[\left(\frac{2+U_0}{R_1}\right)^{\frac{1}{n-1}} - 1\right] + \frac{2[R_1-(1+2U_0)]}{2+U_0} & \text{if } \left(2 - 3\frac{U_0}{R_1}\right)^{\frac{n-1}{n}} > \frac{(2+U_0)}{R_1} \end{cases}. \tag{38}$$

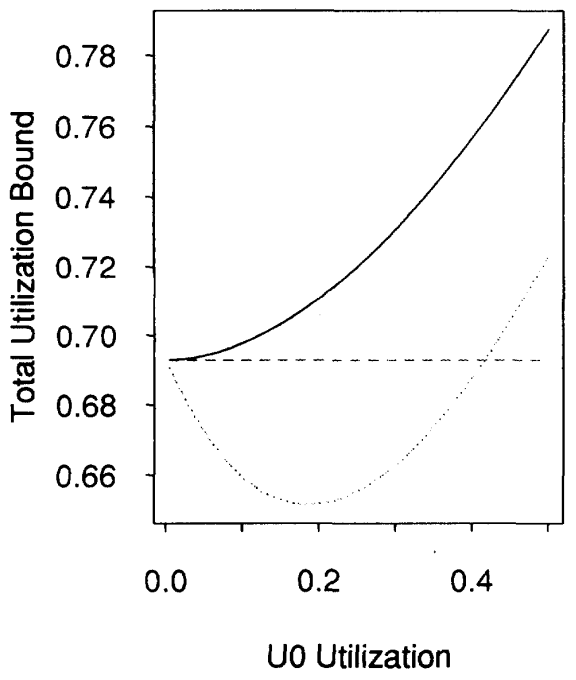


Fig. 10. Case 2 bound. Solid: polling; dotted: DS; dashed: L&L.

Case 3 Bounds: The final case we consider is when the periodic tasks have periods which are both smaller and larger than $T_0 + C_0$, the time at which the DS task runs for the third time. Specifically, we assume there are periodic tasks which satisfy $T_0 \leq T_1 < \cdots < T_k \leq T_0 + C_0 < T_0 + 2C_0 \leq T_{k+1} < \cdots < T_n < 2T_0 + C_0$. The goal is to find the task set with minimum utilization that fully utilizes the processor during $[0, T_n]$ given (C_0, T_0). Once again, a small modification of Lemma B.1 gives a proof that the C_i for the tasks of the worst case task set are given by

$$C_i = T_{i+1} - T_i, \quad 1 \leq i \leq k-1 \tag{42}$$
$$C_k = T_{k+1} - T_k - C_0 \tag{43}$$
$$C_i = T_{i+1} - T_i, \quad k+1 \leq i \leq n-1 \tag{44}$$
$$C_n = 2T_1 - T_n - C_0. \tag{45}$$

The total periodic utilization is given by

$$U_{\text{per}} = \sum_{i=1}^{k-1} \frac{T_{i+1} - T_i}{T_i} + \frac{T_{k+1} - T_k - C_0}{T_k} + \sum_{i=k+1}^{n-1} \frac{T_{i+1} - T_i}{T_i} + \frac{2T_1 - T_n - C_0}{T_n} \tag{46}$$
$$= \sum_{i=2}^{n} R_i - \frac{U_0}{R_1 \cdots R_k} + \frac{2R_1 - U_0}{R_1 \cdots R_n} - n \tag{47}$$

subject to $R_1 \cdots R_k \leq 1 + U_0, R_1 \cdots R_{k+1} \geq 1 + 2U_0$, $R_1 \cdots R_m \leq 2 + U_0$. We first let $S = R_2 \cdots R_k, 1 \leq S \leq (1 + U_0)/R_1$. We seek to minimize

$$\sum_{i=2}^{k} R_i - \frac{U_0}{R_1 S} + \frac{2R_1 - U_0}{R_1 S(R_{k+1} \cdots R_n)} + \sum_{i=k+1}^{n} R_i - n. \tag{48}$$

The minimizing values of R_i are given by $S^{1/(k-1)}, 2 \leq i \leq k$. Thus our objective becomes the minimization of

$$(k-1)S^{1/(k-1)} - \frac{U_0}{R_1 S} + \frac{(2R_1 - U_0)/(R_1 S R_{k+1})}{R_{k+2} \cdots R_n} + \sum_{i=k+2}^{n} R_i + R_{k+1} - n. \tag{49}$$

Lemma A.1 allows us to minimize over $R_{k+2}, \cdots, R_n$. Equation (49) reduces to

$$(k-1)S^{1/(k-1)} - \frac{U_0}{R_1 S} + (n-k)\left\{\frac{2R_1 - U_0}{R_1 S R_{k+1}}\right\}^{1/(n-k)} + R_{k+1} - n, \tag{50}$$

subject to $1 \leq S \leq (1 + U_0)/R_1, R_{k+1} \geq (1 + 2U_0)/R_1 S$. Minimization over R_{k+1} reduces (50) to

$$(k-1)S^{1/(k-1)} - \frac{U_0}{R_1 S} + (n-k+1) \cdot \left\{\frac{2R_1 - U_0}{1 + 2U_0}\right\}^{1/(n-k+1)} + \frac{1 + 2U_0}{R_1 S} - n. \tag{51}$$

At this point, it is convenient to allow n to be large and produce the least upper bound as a function of R_1 and S rather than minimizing over the discrete variable k. Taking limits, we find

$$\text{DS}_{\text{per},\infty}(U_0, R_1) = \ln S + \frac{1 + U_0}{R_1 S} - 1 + \ln\left(\frac{2R_1 - U_0}{1 + 2U_0}\right), \tag{52}$$

subject to $1 \leq S \leq (1 + U_0)/R_1, 1 \leq R_1 \leq 1 + U_0$. This is minimized by taking S as large as possible, that is $S = (1 + U_0)/R_1$. This gives an expression for the least upper bound as a function of U_0 and R_1

$$\text{DS}_{\text{per},\infty}(U_0, R_1) = \ln\left(\frac{1 + U_0}{R_1}\right) + 1 - 1 + \ln\left(\frac{2R_1 - U_0}{1 + 2U_0}\right) \tag{53}$$
$$= \ln\left\{\frac{(1 + U_0)}{(1 + 2U_0)}\left[2 - \frac{U_0}{R_1}\right]\right\}. \tag{54}$$

Thus

$$\text{DS}_{\text{per},\infty}(U_0, R_1) = \ln\left\{\frac{(1 + U_0)}{(1 + 2U_0)}\left[2 - \frac{U_0}{R_1}\right]\right\} \quad \text{for } 1 \leq R_1 \leq 1 + U_0 \text{ and } 0 \leq U_0 \leq 1/2. \tag{55}$$

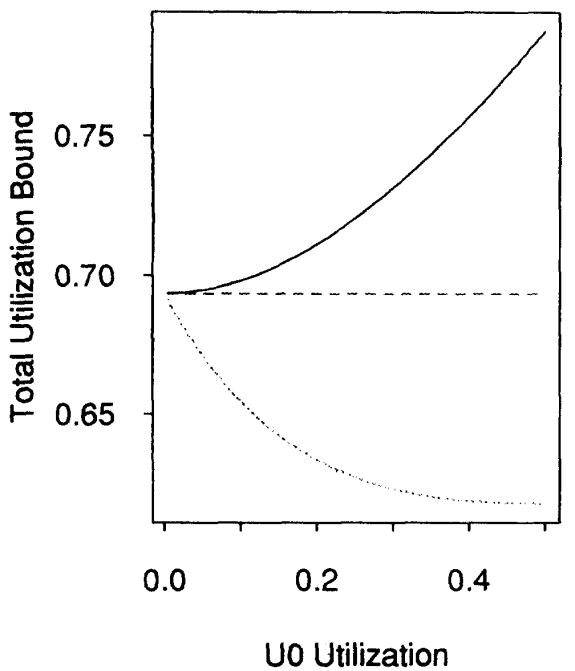


Fig. 11. Case 3 bound. Solid: polling; dotted: DS; dashed: L&L.

The minimizing choice of R_1 is given by $R_2 = 1$. This results in a least upper bound for the periodic component given by

$$\mathrm{DS}_{\mathrm{per},\infty}(U_0) = \ln\left\{\frac{(1+U_0)}{(1+2U_0)}[2-U_0]\right\} \quad \text{for } 0 \le U_0 \le 1/2. \tag{56}$$

The total schedulable utilization in the worst case is given by

$$\mathrm{DS}_{\mathrm{per},\infty}(U_0) = U_0 + \ln\left\{\frac{(1+U_0)}{(1+2U_0)}[2-U_0]\right\}. \tag{57}$$

This bound is plotted in Fig. 11 along with the generalized Liu and Layland bound and the PS bound. It is a decreasing function of U_0 for $U_0 \in [0, 1/2]$. If $U_0 = 0$, the bound is $\ln 2 = 0.693$, the Liu and Layland bound. If $U_0 = 1/2$, the bound is $1/2 + \ln(9/8) = 0.618$.

We complete the analysis by computing the worst case utilization for the situation in which $T_n = 2(C_0+T_0)$. Because $T_n/T_1 < 2$, we assume $T_0 < C_0 + T_0 < T_1 < \cdots < T_{n-1} < T_n = 2(C_0 + T_0)$.

It is obvious that we can reduce consideration to the case $T_0 < 2C_0 + T_0 \le T_1 < \cdots < T_{n-1} < C_0 + 2T_0 < 2(C_0 + T_0) = T_n$. For this case, the minimum utilization is achieved when

$$\begin{aligned} C_i &= T_{i+1} - T_i, \quad 1 \le i \le n-2 \\ C_{n-1} &= C_0 + 2T_0 - T_{n-1} \\ C_n &= 2T_1 = 4C_0 - 2T_0. \end{aligned}$$

The resulting utilization is given by

$$U_{\mathrm{per}} = \sum_{i=2}^{n-1}(R_i - 1) + (C_0 + 2T_0 - T_{n-1})/T_{n-1} + (2T_1 - 4C_0 - 2T_0)/T_n,$$

where $R_i = T_i/T_{i-1}$ and $T_n = 2(C_0 + T_0)$. Standard minimization arguments lead to

$$\mathrm{DS}_{\mathrm{per},n}(U_0, R_1) = (n-2)\left(\left(\frac{U_0+2}{R_1}\right)^{1/(n-2)} - 1\right) + \frac{R_1 - 2C_0 - 1}{U_0 + 1}, \quad 2U_0 + 1 \le R_1 < (U_0 + 2)$$

which is minimized by setting $R_1 = 2U_0 + 1$. Thus, the minimum utilization for this case is

$$\mathrm{DS}_{\mathrm{per},n}(U_0) = (n-2)\left(\left(\frac{U_0+2}{2U_0+1}\right)^{1/(n-2)} - 1\right).$$

or as $n \to \infty$

$$\mathrm{DS}_{\mathrm{per},\infty}(U_0) = \ell n\left(\frac{U_0+2}{2U_0+1}\right).$$

Direct comparison with the results for the three cases considered before show these bounds to be identical to the Case 2 bounds (see Eq. 40) and uniformly larger than those for Case 3 (see Eq. 57). Consequently, $T_n = 2C_0 + 2T_0$ cannot correspond to the worst case. This shows that $T_n/T_0 < 2 + U_0$ is a necessary condition for a worst case task set.

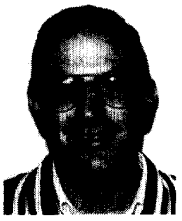
ACKNOWLEDGMENT

The authors wish to thank the referees for a very careful reading of an earlier version of this paper which lead to a substantial improvement in the paper. They also thank Brinkley Sprunt for carrying out the extensive simulation study presented in this paper, and C. J. Paul for his help in the production of this paper.

REFERENCES

[1] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proc. 11th IEEE Real-Time Syst. Symp.*, 1990, pp. 201–209.
[2] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *JACM*, vol. 20, no. 1, pp. 46–61, 1973.
[3] J. Liu, C. L. Liu, and L. Liestman, "Scheduling with slack time," *Acta Informatica*, vol. 17, pp. 31–41, 1982.
[4] J. Y. Leung and M. L. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Inform. Processing Lett.*, vol. 11, no. 3, pp. 115–118, Nov. 1980.
[5] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiprocessors," *Inform. Processing Lett.*, vol. 12, no. 1, pp. 9–12, Feb. 1981.
[6] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proc. 10th IEEE Real-Time Syst. Symp.*, 1989, pp. 166–171.

[7] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda, "Fixed priority scheduling theory for hard real-time systems," in *Foundations of Real-Time Computing: Scheduling and Resource Management,* A.M. van Tilborg and G. M. Koob, Eds. New York: Kluwer Academic, 1991, pp. 1–30.
[8] A. K. Mok, "Fundamental Design problems of distributed systems for the hard real-time environment," Ph.D. thesis, M.I.T., 1983.
[9] L. Sha, J. P. Lehoczky, and R. Rajkumar, "Solutions for some practical problems in prioritized preemptive scheduling,"in *Proc. 7th IEEE Real-Time Syst. Symp.*, 1986, pp. 181–191.
[10] J. K. Strosnider and T. Marchok, "Responsive, deterministic ieee 802.5 token ring scheduling," *Real-Time Syst. J.*, pp. 133–158, Sept. 1989.
[11] B. Sprunt, "Aperiodic task scheduling for real-time systems," Ph.D. thesis, Carnegie Mellon Univ., Aug. 1990.
[12] J. K. Stronsnider, "Highly responsive real-time token rings," Ph.D. thesis, Carnegie Mellon Univ., Aug. 1988.
[13] ——, "Enhanced responsiveness protocol for real-time token rings," Tech. Rep. CMU-90-1, Carnegie Mellon Univ., Jan. 1990.
[14] M. Joseph and P. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, no. 5, pp. 390–394, 1986.



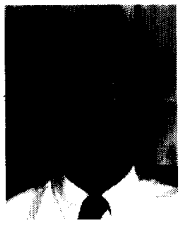
Jay K. Strosnider (M'88) received the B.S. degree in electrical engineering and biotechnology in 1977, and the M.S. degree in electrical engineering in 1978, and the Ph.D. degree in computer engineering in 1988; all from Carnegie Mellon University, Pittsburgh, PA.

He is an Associate Professor in the Electrical and Computer Engineering Department at Carnegie Mellon University. He worked for IBM developing distributed real-time systems for submarines for 1978 to 1985 when he started his Ph.D. studies. He joined the CMU faculty as an Assistant Professor in 1988, and was promoted to Associate Professor in 1993. His current research focus is upon integrating wide ranging technologies within a scheduling theoretic framework. Within the application domains of real-time/multimedia systems, his research interests include: operating systems, networks, buses, disks, window managers fault-tolerance and AI.

John P. Lehoczky (M'88) received the B.A. degree in mathematics from Oberlin College, Oberlin, OH, in 1965, and the M.S. and Ph.D. degrees in Statistics from Stanford University, Stanford, CA, in 1967 and 1969, respectively.

He was an Assistant Professor of Statistics at Carnegie Mellon University, Pittsburgh, PA, from 1969 to 1973, Associate Professor from 1973 to 1981, and Professor from 1981 to the present. He has served as Head of the Department of Statistics since 1984. His research interests involve applied probability theory with emphasis on models in the area of computer and communication systems. In addition, he is the senior member of the Advanced Real-Time Technology (ART) Project in the Carnegie Mellon University School of Computer Science and is doing research in distributed real-time systems, real-time scheduling and fault tolerance.

Dr. Lehoczky is a member of Phi Beta Kappa, a Fellow of the Institute of Mathematical Statistics and the American Statistical Association and is an elected member of the International Statistical Institute. He is a member of the ACM, Operations Research Society of America, and The Institute of Management Science. He served as area editor of @i(Management Science) from 1981 to 1986 and is Associate Editor of @i(Real-Time Systems).



Lui Sha (S'76–M'80–S'81–M'82–SM'91) is a senior member of the technical staff of the Software Engineering Institute. He leads the development and transition of distributed real-time fault-tolerant computing technology at the SEI.

He is an Associate Editor of the international journal, *Real-Time Systems* and an Area Editor of *IEEE Computer*. He has published over 30 articles on real-time computing and served as Chairs and Vice Chairs on several national and international conferences, workshops, and standard working groups. He also served in NASA's Space Station Advisory Committee.