EVALUATION OF THE PROBABILITY OF DYNAMIC FAILURE AND PROCESSOR UTILIZATION FOR REAL-TIME SYSTEMS*

Michael H. Woodbury Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

ABSTRACT

For any real-time system application, given a scheduling policy and task execution time distributions, we show how to determine closed form expressions for task scheduling delay and *active task time* distributions. The active task time denotes the total time a task is executing or waiting to be executed, including scheduling delays and resource contention delays. The distributions are used to determine the *probability of dynamic failure* and processor utilization, where the probability of dynamic failure is the probability that any task will not complete before its deadline. The opposing effects of decreasing the probability of dynamic failure and increasing utilization are also addressed.

The analysis first addresses workloads where all tasks are periodic, i.e., they are repetitively triggered at constant frequencies. It is then extended to include the arrival of asynchronously triggered tasks. The effects of asynchronous tasks on the probability of dynamic failure and utilization are then addressed.

Index Terms — Real-time systems, workload modeling, task execution time modeling, real-time task scheduling.

1 INTRODUCTION

One of the challenges of computer system design and analysis is determining a feasible task scheduling policy to meet specific performance or reliability guidelines. Since most scheduling problems are NP-complete [1], heuristic methods have been developed in practice to obtain acceptable solutions. Once a scheduling algorithm has been proposed, however, it must be validated to determine the probability of satisfying the performance and reliability constraints.

Performance and reliability validation is especially important in the analysis of real-time systems, because of their increasing use in critical applications. Such applications include aircraft and spacecraft control, manufacturing process control, and power generation and distribution, where controlling computer failure could result in catastrophic losses. For these systems, a failure could be the result of a physical malfunction or the system not reacting quickly enough. The latter subsumes the former and is termed *dynamic failure* in [2,3]. For a real-time system, the probability of dynamic failure is the probability that any task will not complete before its deadline. The probability of dynamic failure, therefore, can relate the performance effects on reliability for real-time systems.

For a specific real-time application, the probability of dynamic failure, denoted P_{dyn}, depends on task execution time distributions and a given scheduling policy. The workload of real-time systems consist of a fixed set of tasks which are established before the system begins operation. Thus, it is possible to determine the execution time distribution of all tasks expected to execute on the system. Scheduling policies are selected based on optimizing a desired cost function.

Most scheduling algorithm analyses assume constant task execution times [4,5,6,7] or fixed maximum execution times [8]. Others also are restricted to specific classes of scheduling policies, e.g., non-preemptive [9,10] or those where all tasks have identical release or due times [11]. These methods fail in their usefulness to determine P_{dyn}, because actual task execution times are data dependent [12] and, thus, non-deterministic. The probability of dynamic failure is trivially 1 or 0 with constant execution time assumptions. Thus, this result is only useful for conservative worse case studies.

There have also been scheduling studies where execution times are non-deterministic [13,14,15,16], where most assume exponentially distributed execution times. Their analysis parallels that of the deterministic case, except expected execution times replace deterministic execution times. The objective is to optimize the expected value of a cost function, e.g., expected processing time or number of late tasks. This information is less useful than the deterministic case for determining P_{dyn}, because deterministic worse case values are more informative than expected value results.

Previous scheduling work has been directed towards determining optimal scheduling policies for certain cost functions. As argued above, the results are not directly amenable to the calculation of the probability of dynamic failure, except for the most conservative cases. Therefore, given an optimal scheduling policy, a method is needed to determine if the resulting probability of dynamic failure meets the given specification. This paper addresses that need.

Assuming a scheduling policy is given, the first portion of this

*The work reported in this paper was supported in part by NASA under grants NAG 1-296, NAG 1-492, and NGT 23-005-801. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of NASA.

paper expands on previous results to determine the *active task time distribution* for each task [12,17]. The active task time denotes the total time a task is executing or waiting to be executed and is not always identical to task execution time. A task may be aborted, thus, no longer remaining active, before correctly completing execution. Once the active task time distribution is determined, P_{dyn} is calculated. This will determine if a scheduling policy that has been optimized for some aspect is feasible by having a small enough probability of dynamic failure.

A factor that counters the benefits of minimizing P_{dyn} is processor utilization. A higher processor utilization is more cost effective, because more work is accomplished with the same hardware, in less time. However, as P_{dyn} decreases, utilization also decreases. Therefore, the benefits of each measure must be weighed to determine a feasible schedule for a particular application.

The rest of this paper is organized as follows. The next section introduces necessary definitions and states the specific problems to be solved. Section 3 presents the calculation of the scheduling delay distribution for each system task based on a given scheduling policy. This result is then used to determine the active task time probability distribution. Section 4 addresses the calculation of the probability of dynamic failure and processor utilization, and their countering effects. The initial analysis addresses periodically triggered tasks. Section 5 expands the analysis to include asynchronously triggered tasks. The paper concludes with Section 6.

2 DEFINITIONS AND PROBLEM STATEMENT

A major function of a real-time system is the continuous, timely reaction to system state changes and environmental conditions. Thus, most system tasks are executed repetitively with a fixed period. To maintain system stability real-time tasks have corresponding finite deadlines. A task not completing before its deadline results in a dynamic failure. Initially, a workload where all system tasks are periodic is studied. Section 5 then addresses the impact of asynchronously triggered tasks.

All analysis is performed in terms of unit cycles and multiples of major cycles, defined as follows:

Definition 1 *A unit cycle, u, is the greatest common divisor (GCD) of all periodic task periods.*

Definition 2 *A major cycle is the least common multiple (LCM) of all periodic task periods. Let M denote any multiple of major cycles.*

A unit cycle defines the maximum time interval where it can be guaranteed that no periodic task is triggered during the interval, except at the beginning. Specifically, the i^{th} unit cycle is the time interval $(i-1)u \leq t < iu$, where a periodic task triggered during the interval will be at time $(i-1)u$. A major cycle is the maximum time interval where the simultaneous triggering of all periodic tasks occurs only once. By definition, periodic tasks triggered at time iu are also triggered at time $iu + jM$, where i and j are non-negative integers and $0 \leq i < M/u$.

To determine the active task time distribution, it is necessary to analyze the task at the subtask or instruction level. At this level, instruction steps can be considered to have deterministic execution times. With this detail, accounting for scheduling and resource contention delays is equivalent to determining the priority of the instruction steps from different tasks. Thus, the entire analysis is based on analyzing the sequence of task instructions executed.

A *path* through a task is the sequence of instructions executed given specific initial values for all variables. The number of data dependent branch instructions a task contains determines the maximum number of paths. For example, if a task contains one binary branch, it has two execution paths. If a task has two binary branches, it may have three or four paths depending on its structure. Loops are treated as a sequence of instructions followed by a branch to the beginning of the sequence. Because each instruction step has a deterministic execution time, each path through a task has a deterministic total execution time.

Since there are a finite number of paths through a task[1] and paths have deterministic execution times, the probability distribution function (PDF) of *task execution time* is a staircase function. Therefore, for any task j, let Z_j be the task execution time with PDF

$$F_{Z_j}(t) = \sum_{i=1}^{N_{Z_j}} p_i(Z_j) u_1(t - c_i(Z_j)),$$

where N_{Z_j} is the number of paths in task j, $p_i(Z_j)$ is the probability of path i executing, and $c_i(Z_j)$ is the deterministic execution time of path i. The function u_1 is the unit step function, where

$$u_1(t) = \begin{cases} 0 & \text{if } t < 0, \\ 1 & \text{otherwise.} \end{cases}$$

The justification of this representation is based on work in [12,17], where we demonstrate how to calculate $p_i(\cdot)$ and $c_i(\cdot)$ for any real-time task.

Task execution time is the error-free time required for a task to complete, excluding scheduling and resource contention delays and deadline effects. The *active task time* is derived from the task execution time. Initially, when a task is triggered to execute, there may be tasks of higher priority already queued for execution. In this case, the task would incur a scheduling delay. Once it begins execution, the task may also be preempted by the arrival of higher priority tasks, if the scheduling policy allows preemption. Finally, if a task does not complete before its deadline, it is aborted. Therefore, the active task time is the total time from when the task is triggered to begin until it completes execution or reaches its deadline. Task execution time is equivalent to active task time with no scheduling or contention delays, no task preemption, and inconsequential deadlines. Let T_j denote the active task time with PDF

$$F_{T_j}(t) = \sum_{i=1}^{N_{Z_j}} p_i(T_j) u_1(t - c_i(T_j)).$$

All derivations assume an error-free environment. The execution time of a task is independent of the execution time of all other tasks [4,18]. However, the active task time includes effects of tasks scheduled on the same processor. To simplify notation, context switching time is assumed to be included in the task execution time. (Note that some researchers assume context switching time is zero for task level analyses [5].)

[1] This is because tasks have finite deadlines and instructions have positive finite execution times.

Input and output communication delays are assumed to be constant, and also included in the task execution time. Other researchers have assumed negligible data transfer times [10]. For many real-time applications data is available when necessary and there is minimal contention for resources. However, if there is contention, the tasks that are able to execute continue and the blocked task will incur a scheduling delay. This is analogous to having a task with higher priority than the active task arrive and preempt the executing task.

The analysis assumes a scheduling policy is given. Any scheduling policy is acceptable, where it can be determined which periodic tasks are triggered at the beginning of each unit cycle and relative task priorities do not change during a unit cycle. Each task is analyzed separately, beginning with the highest priority task triggered in the first unit cycle through the last task triggered during the interval analyzed, namely $[0, M]$. The active task time of the task under study, referred to as the *target task*, depends on the execution time of all tasks triggered from the first unit cycle through the unit cycle containing the target task's deadline.

Assuming the scheduling policy and task execution time PDFs are given, we show in the next section the derivation of closed form expressions for the target task scheduling delay PDF and the active task time PDF. Based on these distributions the probability of dynamic failure is derived. Active task time PDFs are also used for the calculation of processor utilization. As mentioned earlier, the probability of dynamic failure cannot be decreased without affecting processor utilization. Finally, the effects of asynchronously arriving tasks are addressed. Closed form expressions for the probability of dynamic failure under the influence of asynchronous tasks are derived.

3 SCHEDULING DELAY AND ACTIVE TASK TIME PDFS

To determine the scheduling delay PDF for the target task, the analysis begins at a point when the processor is idle at the start of a unit cycle. The start of the first unit cycle is guaranteed to be such a point, and others may exist. The target task scheduling delay is defined to be the time from when the task is triggered until it begins execution.

Suppose tasks j and k are triggered during the same unit cycle, the joint distribution of their combined execution time is

$$F_{Z_j+Z_k}(t) = \sum_{i=1}^{N_{Z_j}N_{Z_k}} p_{\kappa_1}(Z_j)p_{\kappa_2}(Z_k)u_1(t - c_{\kappa_1}(Z_j) - c_{\kappa_2}(Z_k)),$$

where $\kappa_1 = \lceil i/N_{Z_k} \rceil$ and $\kappa_2 = i + N_{Z_k} - N_{Z_k}\lceil i/N_{Z_k} \rceil$. The expression $\lceil x \rceil$ is defined as the smallest integer greater than or equal to x. This equation can be recursively applied to determine the combined distribution for any number of tasks.

Let $U(i)$ be the combined execution time of tasks triggered during the i^{th} unit cycle with higher priority than the target task. Tasks which are required to complete before the target task can be defined to have a higher priority. The PDF of $U(i)$ can be expressed as follows,

$$F_{U(i)}(t) = \sum_{j=1}^{N_{U(i)}} p_j(U(i))u_1(t - c_j(U(i))),$$

where $N_{U(i)}$ is the product of the number of paths in each triggered task. Without loss of generality, $F_{U(i)}(t)$ can be expressed such that $c_j(U(i)) \leq c_{j+1}(U(i))$, $j = 1, \ldots, N_{U(i)} - 1$.

The target task scheduling delay PDF, $F_{SD}(t)$, is derived in two steps. The first step determines the PDF of the remaining execution time, RM, of higher priority tasks triggered before the target task. The second step constructs the scheduling delay PDF by including the effects of higher priority tasks triggered after the target task.

Suppose the target task is triggered at the beginning of unit cycle Y_{tr}, which is at time $(Y_{tr} - 1)u$, and has a hard deadline of t_d. Let Y_{td} be the last unit cycle where higher priority tasks can be triggered before t_d, i.e., $Y_{td} = \lceil \frac{t_d}{u} \rceil$. The remaining execution time PDF, $F_{RM}(t)$, is constructed by recursively determining the remaining execution time for each unit cycle through cycle Y_{tr}. Let $F_{RM(i)}(t)$ represent the remaining execution time PDF at the beginning of unit cycle i.

Initially, $F_{RM(1)}(t) = F_{U(1)}(t)$. To derive $F_{RM(i+1)}(t)$, $1 \leq i \leq Y_{tr} - 1$, express $F_{RM(i)}(t)$ as

$$F_{RM(i)}(t) = \sum_{j=1}^{N_{RM(i)}} p_j(RM(i))u_1(t - c_j(RM(i))),$$

such that $c_j(RM(i)) \leq c_{j+1}(RM(i))$, $j = 1, \ldots, N_{RM(i)} - 1$. Initially, $N_{RM(1)} = N_{U(1)}$, and is recursively defined below for $i \geq 2$. If $F_{RM(i)}(u) = 1$,

$$F_{RM(i+1)}(t) = F_{U(i+1)}(t).$$

Otherwise, let $r = \min\{j : u \leq c_j(RM(i))\} - 1$. Then

$$F_{RM(i+1)}(t) =$$

$$\sum_{j=1}^{(N_{RM(i)}-r)N_{U(i+1)}} \frac{p_{\kappa_3}(RM(i))p_{\kappa_4}(U(i+1))}{1 - F_{RM(i)}(u)} u_1(t + \tau_1),$$

where $\tau_1 = u - c_{\kappa_3}(RM(i)) - c_{\kappa_4}(U(i+1))$, $\kappa_3 = r + \lceil j/N_{U(i+1)} \rceil$, and $\kappa_4 = j + N_{U(i+1)} - N_{U(i+1)}\lceil j/N_{U(i+1)} \rceil$. Therefore, the remaining execution time PDF of higher priority tasks triggered before the target task is

$$F_{RM}(t) = F_{RM(Y_{TR})}(t).$$

N_{RM}, the number of steps in the RM PDF, depends on the intermediate PDFs for each $RM(i)$.

The scheduling delay PDF is the combination of the remaining execution time of previously triggered higher priority tasks and the execution time of higher priority tasks triggered after the target task. $F_{SD}(t)$ is recursively derived from unit cycle Y_{tr} through cycle Y_{td}, where $F_{SD(i)}(t)$ is the scheduling delay PDF determined through cycle $i + Y_{tr} - 1$. Any task triggered after t_d will not effect the target task scheduling delay. The algorithm to determine $F_{SD}(t)$ is as follows:

Step 1: (Initialization) Set $i = 1$, where i is a unit cycle index. Let $F_{SD(1)}(t) = F_{RM}(t)$, and $N_{SD(1)} = N_{RM}$. If $F_{SD(1)}(u) = 1$, go to step 3.

Step 2: (Recursion) To derive $F_{SD(i+1)}(t)$, express $F_{SD(i)}(t)$ as

$$F_{SD(i)}(t) = \sum_{j=1}^{N_{SD(i)}} p_j(SD(i))u_1(t - c_j(SD(i))),$$

such that $c_j(SD(i)) \leq c_{j+1}(SD(i))$, $j = 1, \ldots, N_{SD(i)} - 1$. Let $r^* = \min\{j : iu \leq c_j(SD(i))\} - 1$. Then

$$F_{SD(i+1)}(t) = F_{SD(i)}(iu) +$$

$$\sum_{j=1}^{(N_{SD(i)}-r^*)N_{U(Y_{tr}+i)}} p_{\kappa_5}(SD(i))p_{\kappa_6}(U(Y_{tr}+i))u_1(t-\tau_2),$$

where $\tau_2 = c_{\kappa_5}(SD(i)) - c_{\kappa_6}(U(Y_{tr}+i))$, $\kappa_5 = r^* + \lceil j/N_{U(Y_{tr}+i)} \rceil$, and $\kappa_6 = j + N_{U(Y_{tr}+i)} - N_{U(Y_{tr}+i)}\lceil j/N_{U(Y_{tr}+i)} \rceil$. Increment i by one. If $F_{SD(i)}(iu) = 1$ or $i > Y_{td} - Y_{tr}$, go to step 3. Otherwise, repeat step 2.

Step 3: (Assignment) The scheduling delay PDF for the target task is

$$F_{SD}(t) = F_{SD(i)}(t).$$

For use in determining the active task time of the target task, let $Y_{cy} = i + Y_{tr} - 1$. Y_{cy} is the last unit cycle analyzed in determining the scheduling delay of the target task. ■

The calculation of $F_{SD}(t)$ is independent of the scheduling algorithm, and most notably, independent of preemptive or non-preemptive scheduling. All that is required is the capability to determine which tasks must be completed before the target task.

The active task time PDF is recursively derived from cycle Y_{cy} through cycle Y_{td} (If $Y_{cy} = Y_{td}$, only one cycle is analyzed). $F_{T(i)}(t)$ is the active task time PDF analyzed up to cycle $i + Y_{cy} - 1$. For $i = 1$,

$$F_{T(1)}(t) = F_{SD+Z}(t) =$$

$$\sum_{i=1}^{N_{SD}N_Z} p_{\kappa_7}(SD)p_{\kappa_8}(Z)u_1(t - c_{\kappa_7}(SD) - c_{\kappa_8}(Z)),$$

where $\kappa_7 = \lceil i/N_Z \rceil$ and $\kappa_8 = i + N_Z - N_Z\lceil i/N_Z \rceil$.

If $Y_{cy} = Y_{td}$ or the scheduling algorithm is non-preemptive, $F_T(t) = F_{T(1)}(t)$. Otherwise, to derive $F_{T(i+1)}(t)$, $1 \leq i \leq Y_{td} - Y_{cy}$, express $F_{T(i)}(t)$ as

$$F_{T(i)}(t) = \sum_{j=1}^{N_{T(i)}} p_j(T(i))u_1(t - c_j(T(i))),$$

such that $c_j(T(i)) \leq c_{j+1}(T(i))$, $j = 1, \ldots, N_{T(i)} - 1$. Let $\hat{r} = \min\{j : (Y_{cy} - Y_{tr} + i)u \leq c_j(T(i))\} - 1$. Then

$$F_{T(i+1)}(t) = F_{T(i)}((Y_{cy} - Y_{tr} + i)u) +$$

$$\sum_{j=1}^{(N_{T(i)}-\hat{r})N_{U(Y_{cy}+i)}} p_{\kappa_9}(T(i))p_{\kappa_{10}}(U(Y_{cy}+i))u_1(t-\tau_3),$$

where $\tau_3 = c_{\kappa_9}(T(i)) - c_{\kappa_{10}}(U(Y_{cy}+1))$, $\kappa_9 = \hat{r} + \lceil j/N_{U(Y_{cy}+i)} \rceil$, and $\kappa_{10} = j + N_{U(Y_{cy}+i)} - N_{U(Y_{cy}+i)}\lceil j/N_{U(Y_{cy}+i)} \rceil$. Finally,

$$F_T(t) = F_{T(Y_{td}-Y_{cy}+1)}(t).$$

4 P_{dyn} AND PROCESSOR UTILIZATION

Let $T_j(i)$ be the active task time of the j^{th} task triggered at the beginning of unit cycle i. The probability of dynamic failure, P_{dyn}, for the bounded time interval $[0, M]$, is the probability that any task triggered during the interval will not complete before its deadline. If $t_d(T_j(i))$ is the deadline for task $T_j(i)$, then

$$P_{dyn} = 1 - \prod_{i=1}^{\lceil M/u \rceil} \prod_j F_{T_j(i)}\left(t_d(T_j(i))\right).$$

Recall that the active task time PDF is relative to the task triggering time. To determine processor utilization, it is necessary to express all active task time distributions in a common time frame. Let $A_j(i)$ be the *absolute active task time*, where $A_j(i) = (i-1)u + T_j(i)$ and has the PDF

$$F_{A_j(i)}(t) = \begin{cases} 0 & \text{if } 0 \leq t < (i-1)u, \\ F_{T_j(i)}(t - (i-1)u) & \text{otherwise.} \end{cases}$$

Given $F_{A_j(i)}(t)$, the probability that the processor is idle at time t, $L(t)$, is

$$L(t) = \prod_{i=1}^{\lfloor t/u \rfloor} \prod_j F_{A_j(i)}(t).$$

This is equivalent to the probability that the lowest priority active task at the beginning of unit cycle $\lfloor t/u \rfloor$ has completed before t. A necessary condition for periodic task scheduling policies is that $L(t) = 1$ at the end of a major frame. A scheduling policy with $L(t) = 1$ at the end of a major frame is termed a *periodic feasible schedule*. If $L(t) \neq 1$, the processor is unable to handle the periodic task load, guaranteeing that at least one task eventually will miss its deadline. Therefore, if $L(t) \neq 1$ at the end of a major frame, $P_{dyn} = 1$.

The processor utilization at time t, $UT(t)$, is defined to be the fraction of time the processor was executing tasks during the interval $[0, t]$ [4]. In terms of the processor idle time,

$$UT(t) = \left[1 - \frac{\int_0^t L(x)dx}{t}\right] \times 100\%.$$

Therefore, the processor utilization for the interval $[0, M]$ is $UT(M)$.

For a fixed set of tasks and given schedule, varying the length of the unit cycle (which implies proportional changes in the task triggering periods) will affect the probability of dynamic failure and processor utilization. A longer unit cycle reduces the probability of dynamic failure by allowing more time for tasks to complete before others are triggered.[2] However, this also reduces processor utilization. The inverse is also true, where a shorter unit cycle increases both utilization and the probability of dynamic failure.

A parametric plot of P_{dyn} versus utilization can be produced to demonstrate the effects of altering the unit cycle length. For example, Figure 1 shows three possible plots. Plots A and C assume task deadlines are a function of the unit cycle length.

[2] This is always true if task deadlines are dependent on unit cycle length. If deadlines are independent, there may be a minimum P_{dyn} where increasing the cycle length has no effect.

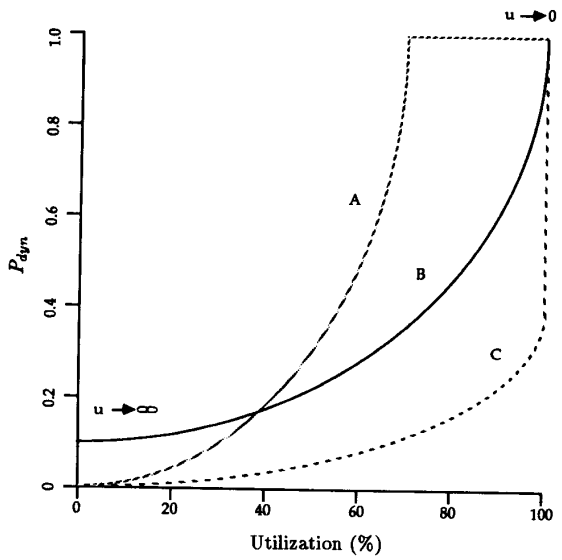

Figure 1: Unit Cycle Effects on P_{dyn} and Utilization

For these cases, P_{dyn} and utilization approach 0 as u increases. It should be noted, however, that it is not necessary to have deadlines be a function of unit cycle length for $P_{dyn} \rightarrow 0$ as $u \rightarrow \infty$. Plot A is a case where $P_{dyn} = 1$ before 100% utilization is attained. This occurs when the unit cycle is short enough to guarantee that a task will miss its deadline. If active task times and deadlines for all tasks are multiples of each other, a trace like plot C is possible. In this situation, 100% utilization is attained before $P_{dyn} = 1$. Finally, if a single task deadline is independent of cycle length, it may not be possible to achieve $P_{dyn} = 0$ as the unit cycle is increased. A plot similar to plot B would then result.

Independent of active task times and deadlines of tasks, all parametric P_{dyn}/utilization plots demonstrate that utilization approaches 100% and $P_{dyn} \rightarrow 1$ as $u \rightarrow 0$. Also, as u increases utilization approaches 0%. Plots derived from experimental results are shown in the next section.

In the design of real-time systems, two important parameters are the maximum probability of dynamic failure and a minimum utilization. The P_{dyn}/utilization plot can be used to determine whether a specific schedule satisfies these criteria and identifies the feasible unit cycle lengths. For some applications, one criterion is more critical. The design goal is to satisfy the less critical criterion while guaranteeing adherence of the more critical one. For example, the probability of dynamic failure is important for flight control systems. It is imperative for a control system to remain below a maximum P_{dyn}. Utilization is a lesser concern. However, for many industrial control applications utilization is as important as P_{dyn}, because of cost considerations. For these systems a design should remain below a maximum P_{dyn} for reliability reasons and maximize utilization to reduce costs.

5 EFFECTS OF ASYNCHRONOUS TASK ARRIVALS

In addition to periodic tasks, real-time systems must allow for the possibility of asynchronously triggered tasks.[3] Any task that is not periodic is considered to be asynchronous. For this analysis, it is assumed that asynchronous tasks have higher priority than the target task. Obviously, the case where an asynchronous task has lower priority than the target task is identical to the situation analyzed earlier. The random arrival of tasks, thus, has the potential of increasing the the probability of dynamic failure.

Asynchronous tasks are triggered through random effects, such as human input, environmental status, or special internal conditions. For this reason asynchronous task arrivals are assumed to be Poisson distributed with rate λ. If there are multiple asynchronous tasks with different active task time PDFs, the exact measurement of the probability of dynamic failure must handle the arrival of each type of task. Let $T_j(a)$ be the active task time of the j^{th} asynchronously triggered task.

To simplify notation, all asynchronous tasks are assumed to have the same PDF which is an upper bound of all asynchronous task PDFs. Specifically, let $T_{max}(a)$ be the active task time of a pseudo asynchronous task representing the maximum active time of all asynchronous tasks, i.e., $T_{max}(a) = \max_j T_j(a)$. Then

$$F_{T_{max}(a)}(t) = \prod_j F_{T_j(a)}(t).$$

The P_{dyn} calculated in this manner is an upper bound of the true probability. Since, P_{dyn} constraints are defined as a maximum value, determining only an upper bound is acceptable. This analysis can be easily extended to account for the individual PDFs of all asynchronous tasks. However, the complexity of the evaluation is increased. The exact calculation of P_{dyn} would then be possible.

To determine an upper bound of P_{dyn}, namely P^*_{dyn}, for the interval $[0, M]$ in the presence of asynchronous tasks, the asynchronous arrival times that produce the maximum potential scheduling delay must be identified. To achieve this, the following lemma and theorems are necessary.

Lemma 1 *For a periodic feasible schedule, the maximum number of active periodic tasks occurs at the beginning of a major frame.*

Proof: By definition, a major frame is the least common multiple of all task periods. Therefore, all periodic tasks are triggered simultaneously at the beginning of any major frame. $Q.E.D$

Theorem 1 *For a periodic feasible schedule, suppose a single asynchronous task is triggered during unit cycle i. By only varying the arrival time of the asynchronous task, P_{dyn} is maximized when the asynchronous task is triggered at the beginning of the unit cycle.*

[3] For brevity, asynchronously triggered tasks will also be referred to as asynchronous tasks.

Proof: Maximizing P_{dyn} is equivalent to maximizing the preemption delay for each periodic task. Let t_{ar} be the arrival time of the asynchronous task, $(i-1)u \leq t_{ar} < iu$. Periodic tasks can only be triggered at the beginning of a unit cycle, because the unit cycle is the greatest common divisor of all periodic task periods. Therefore, no periodic task is triggered at time t, where $(i-1)u < t < t_{tr}$. Let $P_{dyn}(x)$ be the probability of dynamic failure when the asynchronous task arrives at time x. If the asynchronous task arrives at $t_{ar} - \epsilon$, where $0 < \epsilon \leq t_{tr} - (i-1)u$, no periodic task will complete sooner than if the asynchronous task arrived at t_{ar}. In other words, all periodic tasks are either unaffected or delayed. Therefore, $P_{dyn}(t_{ar} - \epsilon) \geq P_{dyn}(t_{ar})$. P_{dyn} is, therefore, maximized when $\epsilon = t_{tr} - (i-1)u$. $\mathcal{Q.E.D}$

Theorem 2 *For a periodic feasible schedule, suppose n asynchronous tasks are triggered during a major frame. By only varying the arrival time of the n asynchronous tasks, P_{dyn} is maximized when all n periodic tasks arrive simultaneously at the beginning of a major frame.*

Proof: Select one of the n asynchronous tasks. By Theorem 1, for the single asynchronous task, $P_{dyn}((i-1)u) \geq P_{dyn}(t)$ for $(i-1)u \leq t < iu$, where unit cycle i is within the major frame. Therefore, to determine the arrival time within the major frame that generates the maximum P_{dyn} through varying the triggering time of the single asynchronous task, the only times that need to be considered are the beginning of unit cycles. By Lemma 1, the maximum number of periodic tasks are delayed when the arrival is at the beginning of the first unit cycle. Thus, the maximum P_{dyn} that can be generated by a single asynchronous task is when it arrives at the beginning of the major frame. To complete the proof, repeat the process for the other $(n-1)$ asynchronous tasks. $\mathcal{Q.E.D}$

To determine P^*_{dyn} for the interval $[0, M]$, the upper bound for P_{dyn} is calculated for a fixed number, varying from 0 to N_{as} (N_{as} is defined below), of asynchronous tasks. The P_{dyn} values are then weighted by the probability of having the respective number of asynchronous task triggered. Because the arrival process of the asynchronous tasks is Poisson, there is a non-zero probability of having any finite number of tasks triggered during the interval $[0, M]$. This would result in a prohibitive number of tasks for practical analysis. However, N_{as} is the maximum number of asynchronous tasks that actually need to be considered.

Recall from earlier representations of the active task time PDFs, $c_1(T_j(i))$ is the minimum execution time for the j^{th} periodic task triggered during the i^{th} unit cycle. Similarly, let $c_1(T_j(a))$ be the minimum execution time for the j^{th} asynchronous task. N_{as} is the minimum integer such that

$$N_{as} > \frac{M - \sum_{i=1}^{M/u} \sum_{j} c_1(T_j(i))}{\min_j c_1(T_j(a))} .$$

Based on this definition, if N_{as} or more asynchronous tasks arrive during the interval $[0, M]$, at least one task is guaranteed to miss its deadline. This implies $P_{dyn} = 1$ for this case.

Let $N(t)$ equal the number of Poisson arrivals in the bounded interval $[0, t]$. As shown in [19], for any finite n,

$$\Pr[N(t) = n] = e^{-\lambda t} \frac{(\lambda t)^n}{n!}.$$

Table 1: $\Pr[N(M) \geq N_{as}]$ when $N_{as} = 5$

λM	$\Pr[N(M) \geq 5]$
0.1	7.66780e-08
0.01	8.26421e-13
0.001	5.81725e-18
0.0001	1.34871e-18

Let $[P_{dyn} \mid i \text{ asyn. tasks}]$ represent the probability of dynamic failure over the interval $[0, M]$ given that i asynchronous tasks are triggered. The exact value of P_{dyn} over the interval $[0, M]$ now becomes

$$P_{dyn} = \Pr[N(M) \geq N_{as}] + \sum_{i=0}^{N_{as}-1} \Pr[N(M) = i] [P_{dyn} \mid i \text{ asyn. tasks}]$$

$$= 1 - \sum_{i=0}^{N_{as}-1} \Pr[N(M) = i] + e^{-\lambda M} \sum_{i=0}^{N_{as}-1} \frac{(\lambda M)^i}{i!} [P_{dyn} \mid i \text{ asyn. tasks}].$$

Finally, because only the upper bound on P_{dyn} is calculated for each amount of asynchronous task arrivals

$$P^*_{dyn} = 1 - \sum_{i=0}^{N_{as}-1} \Pr[N(M) = i] + e^{-\lambda M} \sum_{i=0}^{N_{as}-1} \frac{(\lambda M)^i}{i!} [\max P_{dyn} \mid i \text{ asyn. tasks}].$$

Regardless of the value of N_{as}, Table 1 shows that for all practical considerations $\Pr[N(M) \geq 5]$ can be considered negligible for typical values of λM.

To determine $[\max P_{dyn} \mid i \text{ asyn. tasks}]$, the active task time PDF for each periodic task is calculated with the additional effects, if any, of i asynchronous tasks. Recall from the previous section that $U(1)$ represents the combined execution time of all tasks triggered during the first unit cycle with higher priority than the target task. Based on Theorem 2, to determine $\max P_{dyn}$, $U(1)$ should include the i asynchronous tasks if the target task has lower priority than asynchronous tasks. The remaining analysis is identical.

To study the effects of unit cycle length on P^*_{dyn} and utilization, a lower bound on utilization is needed. A lower bound on utilization in the presence of asynchronous tasks is the same value determined in the previous section when no asynchronous tasks were considered, namely $UT(M)$. Parametric plots can be produced to evaluate the relative changes of P^*_{dyn} and $UT(M)$ by varying the unit cycle length under the influence of asynchronous tasks.

To illustrate the effects of asynchronous tasks on the probability of dynamic failure and utilization, experiments were performed on the experimental system in the Real-Time Computing Laboratory, The University of Michigan. The computing system

is a Motorola 68080/VME bus based system running the real-time kernel pSOS [20]. Using a synthetic workload generator, seven tasks were created with structures similar to a control task from an aircraft transport and automatic landing simulation.

The synthetic workload consisted of of three tasks with a triggering period of one unit cycle, two tasks with a two unit cycle period, and one task with an eight unit cycle period. A single asynchronous task had an exponentially distributed interarrival duration. During execution, a rate monotonic preemptive scheduling algorithm was implemented with asynchronous tasks given higher priority than periodic tasks. The plots in Figures 2-4 present the results of varying the duration of the unit cycle and the asynchronous task interarrival rate. For all three plots, the points denoted by '+' symbols are for a workload that does not include the single asynchronous task. Workloads with an asynchronous task interarrival rate of 0.05 and 0.025 sec^{-1} are shown by the 'X' and 'Y' symbols, respectively.

Figure 2 illustrates how P_{dyn} increases as the unit cycle length decreases. As expected, triggering asynchronous tasks at faster rates also increases the probability of dynamic failure. A point worth noting is that the probability of an asynchronous task arriving during a particular unit cycle decreases when the unit cycle length is shortened. Thus, the effects of asynchronous tasks are reduced. Figure 3 shows similar results, where shortening the unit cycle length or increasing the asynchronous tasks interarrival time increases utilization.

A parametric plot of P_{dyn} versus utilization as a function of unit cycle length is shown in Figure 4. This plot vividly illustrates the effects of asynchronous task arrivals on the countering effects of the probability of dynamic failure and utilization. If asynchronous tasks are included, one may not be able to achieve a minimum utilization percentage and still have an acceptable probability of dynamic failure. For example, suppose the analysis of a system produces results shown in Figure 4 and a design constraint requires a minimum utilization of 80% with a maximum P_{dyn} of 0.2. The plot shows that the system will not meet the constraints if the asynchronous task interarrival rate is greater that 0.05 sec^{-1}. In this manner, an acceptable range of task interarrival rates can be defined to meet P_{dyn} and utilization constraints.

6 CONCLUSION

Given a scheduling policy and task execution time distributions, we have shown how to calculate closed form expressions for the scheduling delay and active task time PDFs of all tasks. The derivation of task execution time PDFs is based on results in previous work [12,17]. With this information, the probability of dynamic failure for the system and processor utilization was calculated.

Because tasks triggered early in a major cycle have an effect on tasks triggered later, a recursive approach to determining the active task time PDF of all tasks is evident. The first target task analyzed should be the highest priority periodic task triggered during the first unit cycle. The remaining tasks from the first unit cycle are then analyzed sequentially, based on their relative priorities. Each subsequent task analyzed can use the active task time PDFs derived from earlier triggered tasks of lower priority. Although, the methodology presented in this paper allows tasks to be analyzed in any order, if the active task time PDFs of all tasks is desired, this approach will minimize the total calculations required.

In the calculation of P_{dyn}, it was assumed that the probability an asynchronous task misses its deadline is negligible. This is justified, because asynchronous task interarrival times are much longer than their execution times, and usually have higher priority than periodic tasks. A natural extension of this work would account for the possibility of near coincident asynchronous task arrivals.

It should be noted that because of environmental constraints, it may not be possible to alter the duration of the unit cycle. However, deadlines can be varied independent of the unit cycle and have the same effect.

The results of this paper can be used to identify trouble spots in scheduling algorithms, i.e., determine if an "optimal" scheduling algorithm based on some cost function is feasible. As noted in the introduction, most scheduling policies do not account for non-deterministic task execution times. Therefore, when these algorithms are implemented with random execution times, they may be infeasible due to an unacceptably high probability of dynamic failure. This work aids in identifying this problem.

References

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W.H. Freeman and Co., San Francisco, 1979.

[2] C. M. Krishna and K. G. Shin, "Performance Measures for Multiprocessor Controllers", in *Performance '83*, A. K. Agrawala and S. K. Tripathi, Eds., pp. 229–250, North-Holland, New York, 1983.

[3] K. G. Shin, C. M. Krishna, and Y. H. Lee, "A Unified Method for Evaluating Real-Time Computer Controllers and Its Application", *IEEE Trans. on Automatic Control*, vol. AC-30, no. 4, pp. 357–366, April 1985.

[4] 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, January 1973.

[5] S. K. Dhall and C. L. Liu, "On a Real-Time Scheduling Problem", *Operations Research*, vol. 26, no. 1, pp. 127–140, January 1978.

[6] K. Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems", *Computer*, vol. 15, no. 6, pp. 50–56, June 1982.

[7] J.A. Stankovic, K. Ramamritham, and S. Cheng, "Evaluation of a Flexible Task Scheduling Algorithm for Distributed Hard Real-Time Systems", *IEEE Trans. on Computers*, vol. C-34, no. 12, pp. 1130–1143, December 1985.

[8] P.-Y. R. Ma, Y. S. Lee, and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems", *IEEE Trans. on Computers*, vol. C-31, no. 1, pp. 41–47, January 1982.

[9] J. K. Lenstra and A. H. G. R. Kan, "Complexity of Scheduling Under Precedence Constraints", *Operations Research*, vol. 26, no. 1, pp. 23–35, Jan. - Feb. 1978.

[10] H. Kasahara and S. Narita, "Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing", *IEEE Trans. on Computers*, vol. C-33, no. 11, pp. 1023–1029, June 1984.

[11] S. Sahni, "Preemptive Scheduling with Due Dates", *Operations Research*, vol. 27, no. 5, pp. 925–934, Sept. - Oct. 1979.

[12] M. H. Woodbury, "Analysis of the Execution Time of Real-Time Tasks", in *Proc. Real-Time Systems Symposium*, pp. 89–96, December 1986.

[13] J. Bruno, P. Downey, and G. N. Frederickson, "Sequencing Tasks with Exponential Service Times to Minimize the Expected Flow Time or Makespan", *Jour. of the ACM*, vol. 28, pp. 100–113, 1981.

[14] M. Pinedo, "Stochastic Scheduling with Release Dates and Due Dates", *Operations Research*, vol. 31, no. 3, pp. 559–572, May - June 1983.

[15] G. Weiss and M. Pinedo, "Scheduling Tasks with Exponential Service Times on Nonidentical Processors to Minimize Various Cost Functions", *Journal of Applied Probability*, vol. 17, pp. 187–202, 1980.

[16] R. R. Weber, "Scheduling Jobs with Stochastic Processing Requirements on Parallel Machines to Minimize Makespan or Flowtime", *Journal of Applied Probability*, vol. 19, pp. 167–182, 1982.

[17] M. H. Woodbury and K. G. Shin, "Determining the Active Task Time Distribution for Real-Time Systems", *Submitted to IEEE Trans. on Computers.*

[18] A. Thomasian and P.F. Bay, "Analytic Queueing Network Models for Parallel Processing of Task Systems", *IEEE Trans. on Computers*, vol. C-35, no. 12, pp. 1045–1054, December 1986.

[19] S. M. Ross, *Stochastic Processes*, John Wiley & Sons, New York, 1983.

[20] *pSOS-68K Real-Time, Multi-processing Operating System Kernel User's Manual*, Software Components Group, Inc., 1986.

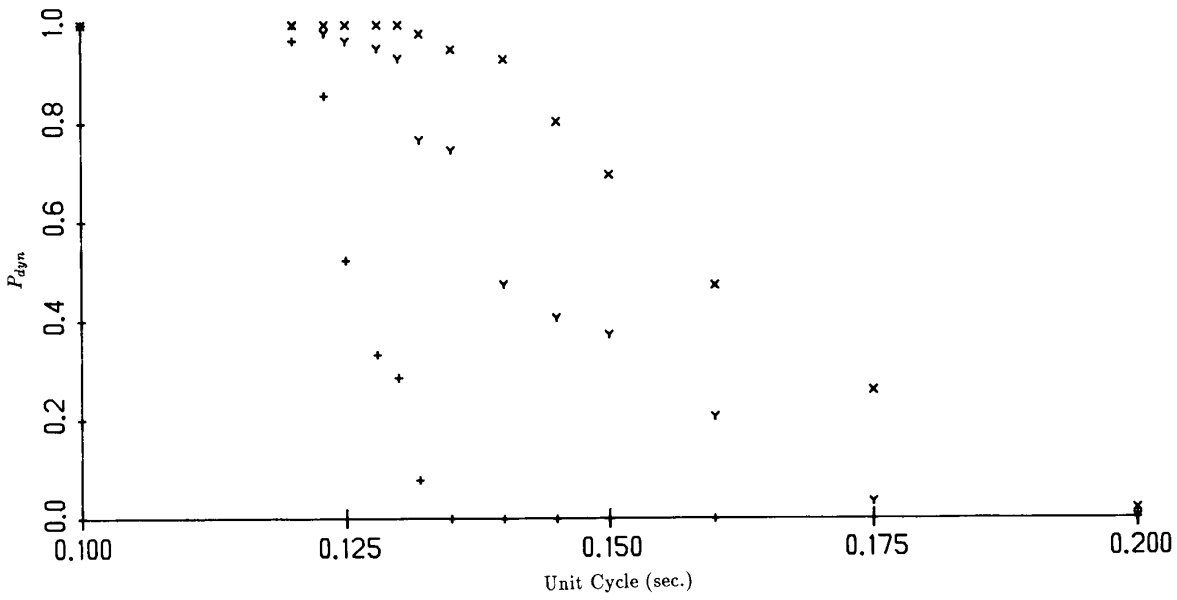

Figure 2: Unit Cycle Effects on P_{dyn}

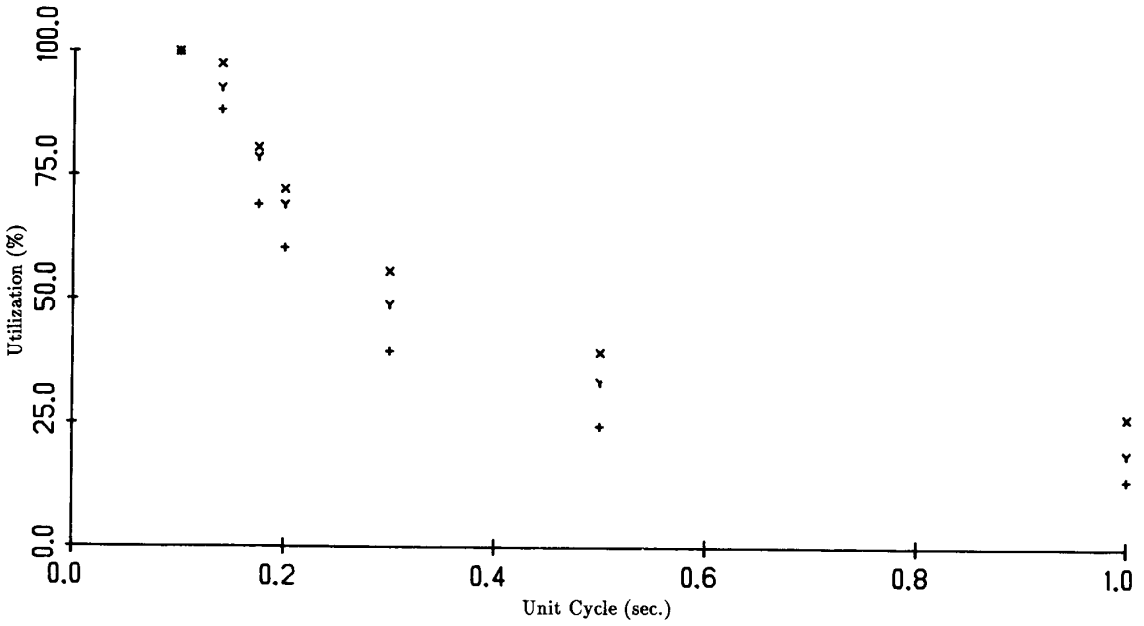

Figure 3: Unit Cycle Effects on Utilization

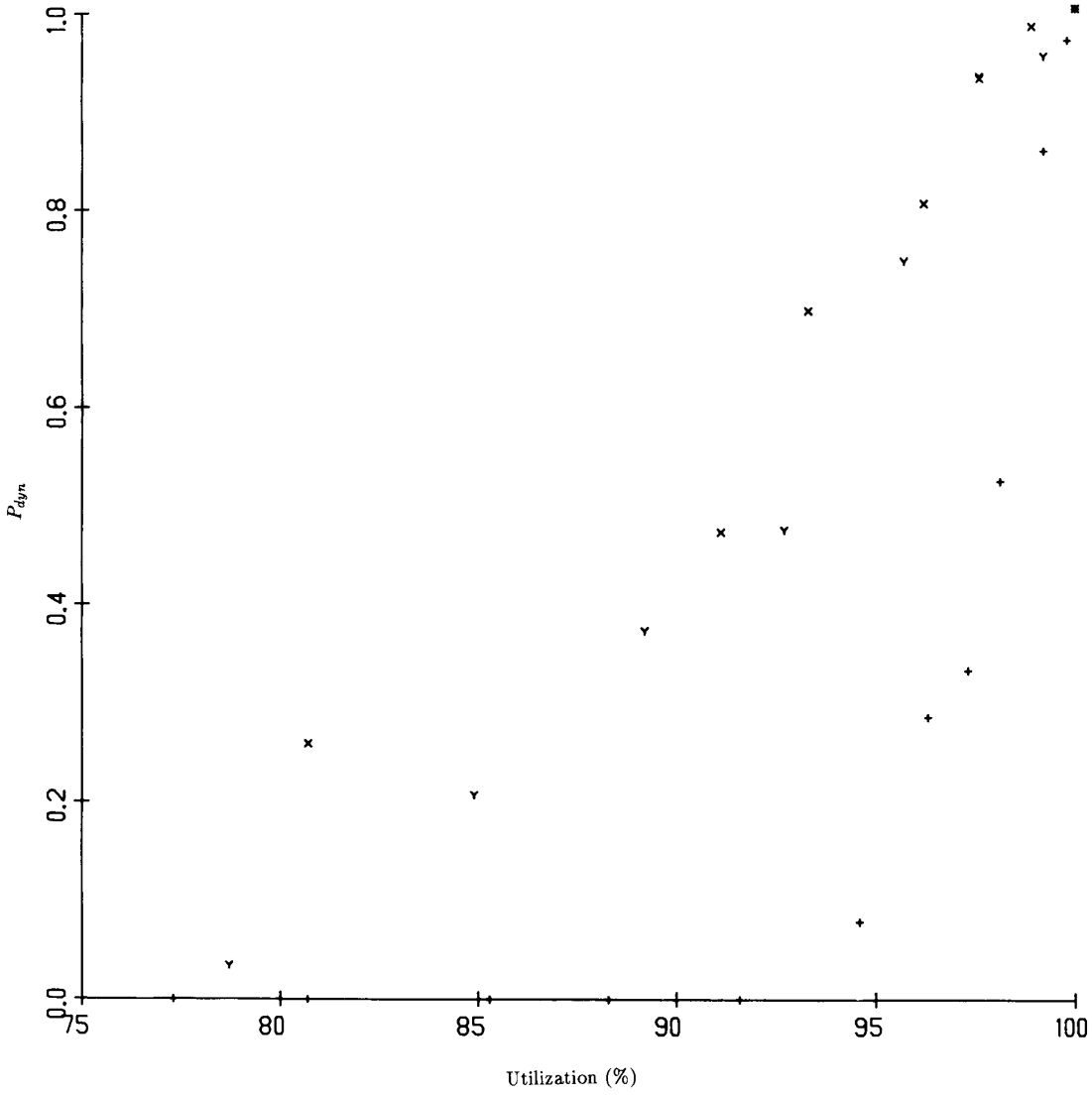

Figure 4: Parametric Plot of P_{dyn} Versus Utilization