In Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 22-26 March 1999, Amsterdam, the Netherlands published as Lecture Notes in Computer Science, vol. 1579, pp. 44-58. Springer-Verlag, Berlin, 1999.

# Analyzing Stochastic Fixed-Priority Real-Time Systems

Mark K. Gardner and Jane W.S. Liu

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{mkgardne,janeliu}@cs.uiuc.edu

**Abstract.** Traditionally, real-time systems require that the deadlines of all jobs be met. For many applications, however, this is an overly stringent requirement. An occasional missed deadline may cause decreased performance but is nevertheless acceptable. We present an analysis technique by which a lower bound on the percentage of deadlines that a periodic task meets is determined and compare the lower bound with simulation results for an example system. We have implemented the technique in the PERTS real-time system prototyping environment [6, 7].

## 1 Introduction

A distinguishing characteristic of real-time computer systems is the requirement that the system meet its temporal constraints. While there are many different types of constraints, the most common form is expressed in terms of deadlines: a job completes its execution by its deadline. In a *hard real-time system*, all jobs must meet their deadlines and a missed deadline is treated as a fatal fault. Hence hard real-time systems are designed to ensure that there are no missed deadlines, often at the expense of resource utilization and average performance. Hard real-time systems are most often found in safety or mission critical applications.

The last few years have seen the proliferation of applications known as *soft real-time systems*. Examples include telecommunications and signal processing systems. For these systems, missed deadlines result in performance degradation. However, provided that the frequency of missed deadlines is below some threshold, the real-time performance of such a system is nevertheless acceptable. While many techniques for designing and validating hard real-time systems exist, there are few such techniques for soft real-time systems. In this paper, we present a schedulability analysis technique for fixed-priority systems to determine lower bounds on the frequency of missed deadlines and compare the lower bound with simulation results for an example system.

We begin, in the next section, with a brief review of schedulability analysis techniques for validating hard real-time systems, motivate the need for better techniques to analyze soft-real time systems and describe closely related work. In Section 3, we present the Stochastic Time Demand Analysis technique and show how it allows a designer greater freedom in trading the certainty with which deadlines are met for other design goals. We compute a lower bound on the probability that deadlines are met by jobs in a simple system and compare the bounds with the percentage of deadlines met obtained by simulation. Section 4 briefly discusses issues which we discovered while implementing STDA in the PERTS real-time system prototyping environment and Section 5 discusses possible directions of future research.

## 2 Background and Related Work

The *periodic task model* [5] has proven useful in describing the characteristics of real-time systems. It is the foundation of state-of-the-art techniques for analyzing the behavior of hard real-time systems. According to this model, a real-time system consists of a set of *tasks*, each of which consists of a (possibly) infinite stream of computations or communications, called *jobs*. We denote the $i$th task of the system by $T_i$ and the $j$th job of the task (or the $j$th job since some time instant) by $J_{i,j}$. The execution time of a job is the amount of time the job takes to complete if it executes alone. All the jobs in a task have a common minimum (maximum) execution time denoted $E_i^-$ ($E_i^+$). Moreover, the jobs are released for execution, (i.e., arrive), with a common minimum inter-release time. The minimum inter-release time (or inter-arrival time) is called the *period* of the task. The period of each task $T_i$ is larger than zero and is denoted by $P_i$. A job $J_{i,j}$ becomes ready for execution at its release time, $r_{i,j}$. It must complete execution by its absolute deadline, $d_{i,j}$, or it is said to have missed its deadline. Figure 1 shows these quantities in the context of a time-line. The length of time between the release time and absolute deadline of every job in each task $T_i$ is constant. This length is called the *relative deadline* of the task and is denoted $D_i = d_{i,j} - r_{i,j}$. The completion time of $J_{i,j}$ is denoted $c_{i,j}$ and the response time is $\rho_{i,j} = c_{i,j} - r_{i,j}$.

In this paper, we will have occasion to refer to the actual execution time of $J_{i,j}$ which we denote $e_{i,j}$. The maximum utilization of the task is the ratio of the maximum execution time to the minimum inter-arrival time (period) and is denoted by $U_i = E_i^+/P_i$. Finally, the release time of the first job in a task is called the *phase* of the task. We say that tasks are *in-phase* when they have identical phases.

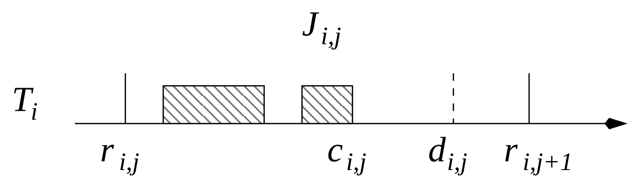



**Fig. 1.** Time-line for Task $T_i$

In modern real-time systems, tasks are scheduled in a priority driven manner. At any point in time, the ready job with the highest priority executes. If at time $t$, a job of a higher priority becomes ready, the executing job is preempted and the higher priority job executes. Most priority assignments are fixed priority. According to a fixed-priority scheduling policy, all jobs in a task have the same priority. We denote the priority of task $T_i$ and hence the priority of jobs $J_{i,1}, J_{i,2}, \ldots$ by $\phi_i$. For convenience and without loss of generality, we assume that priorities are distinct and arrange the tasks in order of decreasing priority $T_i \preceq T_{i+1}$ such that $T_1$ has a higher priority than $T_2$, etc.

### 2.1 Deterministic Schedulability Analysis Methods

A task in a system is said to be *schedulable* if all jobs in the task meet their deadlines. A system of real-time tasks is schedulable if all tasks in the system are schedulable. One of the most commonly used fixed-priority assignments is Rate Monotonic (RM). According to this policy, the shorter the period $P_i$ of a task, the higher its priority. It is shown in [5] that a system of $n$ tasks scheduled on a RM basis is schedulable if the sum of the maximum utilizations of the tasks, denoted $U$, satisfies the inequality $U \leq n(2^{\frac{1}{n}} - 1)$. The expression on the right hand side of the inequality is often called the Liu and Layland bound. The Liu and Layland bound gives a sufficient and hence conservative condition. A system may be schedulable rate monotonically even though its maximum utilization exceeds the Liu and Layland bound.

The Time Demand Analysis (TDA) method [4] provides a more accurate and general characterization of the ability of arbitrary fixed-priority systems to meet all deadlines. It is based upon the observation that the worst-case response time of a job occurs when it is released at a *critical instant*. For a system of independent preemptive periodic tasks scheduled on a fixed-priority basis, a critical instant of a task occurs when a job in each task is released along with a job from all tasks of equal or higher priority [5]. Therefore, to bound the worst case response time of all the jobs in a task $T_i$, it suffices for us to look at a job that is released at a critical instant. We call this job $J_{i,1}$. The *time demand function* of $T_i$, denoted $w_i(t)$, is the total maximum time demanded by $J_{i,1}$, as well as all the jobs that complete before $J_{i,1}$, as a function of time $t$ since the release of $J_{i,1}$. It is a function which increases by the maximum execution time $E_k^+$ every time a higher priority job $J_{k,l}$ is released. If there is a $t \leq D_i$ such that $w_i(t) \leq t$ is satisfied, then no job in $T_i$ will miss its deadline.

Figure 2 shows the time demand function for each of the tasks in Example #1. The parameters of the tasks are listed in Table 1.[1] There is sufficient time for tasks $T_1$, $T_2$ and $T_3$ by 100, 200 and 600 respectively. A schedule of the system with the initial job in each task released at a critical instant is shown in Fig. 3. Even though the processor is idle from 1100–1200, it is clear that increasing the maximum execution time of any task will result in $J_{3,1}$ missing its deadline at 600.

[1] We note that the maximum total utilization of the tasks is 0.92, greater than the Liu and Layland bound, which is 0.78. However, the system is schedulable.

**Table 1.** Parameters of the Tasks in Example #1.

| $T_i$ | $\phi_i$ | $P_i$ | $E_i^+$ | $D_i$ | $U_i$ |
|---|---|---|---|---|---|
| $T_1$ | 1 | 300 | 100 | 300 | 0.333 |
| $T_2$ | 2 | 400 | 100 | 400 | 0.250 |
| $T_3$ | 3 | 600 | 200 | 600 | 0.333 |
| Total | | | | | 0.917 |

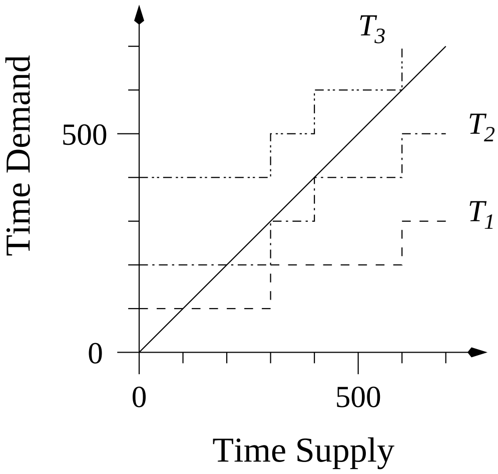



**Fig. 2.** Time Demand Analysis of the Example System

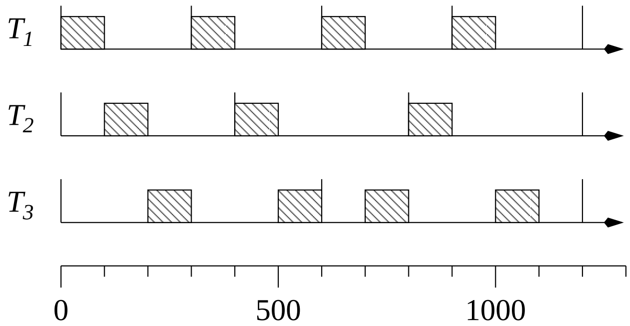



**Fig. 3.** Schedule of the Example System

The version of TDA given above works only when all jobs will complete by the release of the next job in the task, which is the case for the example. To determine whether all jobs in $T_i$ meet their deadlines when some job $J_{k,l+1}$ may be released before the previous job $J_{k,l}$ in a higher priority task $T_k$ completes, we must compute the worst case bound on response times of all jobs in $T_i$ executed in a *level-i busy interval* that begins at an instant when a job $J_{i,1}$ in $T_i$ is released at the same time with a job in every higher priority task.[2] (A level-$i$ busy interval is an interval of time which begins when a job in $T_i$ or a higher priority task is released and immediately prior to the instant no job in those tasks is ready for execution. It ends at the first time instant $t$ at which all jobs in $T_i$ and higher priority tasks released before $t$ have completed.) We call such a busy interval an *in-phase level-$\phi_i$ busy interval.*

Analogous to the critical instant analysis in [5], it has been shown in [3] that it suffices for us to consider only an in-phase level-$\phi_i$ busy interval. The reasons are

1. if a job in $T_i$ is ever released at the same time as a job in every higher priority task, that instant is the beginning of an in-phase busy interval (i.e., the system has no backlog at that instant),
2. the length of an in-phase level-$\phi_i$ busy interval is longer than a level-$\phi_i$ busy interval that is not in-phase (and hence more jobs in $T_i$ are released in the in-phase busy interval), and
3. the response time of every job in a level-$\phi_i$ busy interval that is not in phase is no greater than the response time of the corresponding job in an in-phase level-$\phi_i$ busy interval.

For these reasons, if all jobs in an in-phase level-$i$ busy interval meet their deadlines, the task is schedulable [3]. Stochastic Time Demand Analysis described in Section 3 uses this generalization of TDA.

We know from the above analysis that the system of tasks in Table 1 is schedulable. However, suppose that a significantly less expensive processor is available which is half as fast. The profitability of the product would be greatly enhanced if the slower processor could be used. Using the slower processor, the execution time doubles but the periods do not change because they are determined by the environment. Thus the system utilization is doubled. The hard real-time analysis techniques discussed earlier tell us whether or not a deadline will be missed, but not how often. Although we may be willing to trade occasional missed deadlines for the use of the slower processor, we are unable to do so based on available hard real-time techniques. A different approach is needed.

### 2.2 Probabilistic Approaches

We are aware of only two other techniques that exploit information about the statistical behavior of periodic tasks to facilitate better design of soft real-time

[2] This instant is still called a critical instant in the literature but it is not the original definition of a critical instant since $J_{i,1}$ no longer has the longest response time among all jobs in $T_i$.

systems: Probabilistic Time Demand Analysis (PTDA) [10] and Statistical Rate Monotonic Scheduling (SRMS) [1].

Like the proposed method, PTDA attempts to provide a lower bound on the probability that jobs in a task will complete in time. It is a straight forward extension to TDA in which the time demand is computed by convolving the probability density functions of the execution times instead of summing the maximum execution times as in TDA. PTDA assumes that the relative deadline of all tasks are less than or equal to their periods. It computes a lower bound on the probability that jobs in a task complete in time by determining the probability that the time supply equals or exceeds the time demand at the deadline of the first job in the task. This assumption is not valid, especially when the average utilization of the system approaches one.

SRMS is an extension to classical Rate Monotonic Scheduling (RMS). Its primary goal is to schedule tasks with highly variable execution times in such a way that the portion of the processor time allocated to each task is met on the average. Variable execution times are "smoothed" by aggregating the executions of several jobs in a task and allocating an execution time budget for the aggregate (which may be proportional to the original). A job is released only if its task contains sufficient budget to complete in time and if higher priority jobs will not prevent its timely completion. All other jobs are dropped. The analysis given in [1] can only be used to compute the percentage of jobs in each task that will be released for execution (and hence complete in time). Moreover, it is applicable only when the periods of the tasks are related in a harmonic way, i.e., each larger period $P_j$ is an integer multiple of every smaller period $P_i$. The method presented here seeks to provide a lower bound on the percentage of jobs which meet their deadlines when all jobs are released. It is not restricted to harmonic systems and the RM scheduling policy.

## 3 Stochastic Time Demand Analysis

In this section we describe an algorithm, called Stochastic Time-Demand Analysis (STDA), which computes a lower bound on the probability that jobs in each task will meet their deadlines. We also compare the bound with the average behavior of a system as determined by simulation.

Consider the execution of a task $T_i$. Let $J_{i,j}$ be the $j$th job in $T_i$ released in a level-$\phi_i$ busy interval. To simplify the discussion and without loss of generality, we take as the time origin the beginning of this interval. The response time $\rho_{i,j}$ of job $J_{i,j}$ is a function of the execution times of all jobs which can execute in the interval $[r_{i,j}, c_{i,j})$. As in the deterministic analysis, we use the minimum inter-release time in our analysis. However, the execution times of tasks are random variables, hence the response time of each job in a task is a random variable. Our analysis assumes that the execution time $E_i$ of a job in $T_i$ is statistically independent of that of other jobs in $T_i$ and jobs in other tasks. Again, because a job may not complete by the release of the subsequent job in the same task, we must consider all jobs in a level-$\phi_i$ busy interval, and note that the length of a level-$\phi_i$ busy interval is also a random variable. Bounding the length of a level-$\phi_i$ busy interval is key to STDA. First we show how to compute the response time distribution of jobs in task $T_i$.

Let $w_{i,j}(t)$ denote the time demand of all jobs that execute in the interval $[r_{i,j}, t)$. Job $J_{i,j}$ completes when there is sufficient time to meet the demand $w_{i,j}(t) = t$. Let $W_{i,j}(t) = \mathcal{P}[w_{i,j}(t) \leq t]$ denote the probability that the time demand up to $t$ is met at $t$, given that the busy interval has not ended. We note that $W_{i,j}(t)$ is also the probability that the response time of $J_{i,j}$ is less than or equal to $t$. The probability that $J_{i,j}$ meets its deadline is therefore at least $W_{i,j}(D_i)$. We now turn our attention to computing $W_{i,j}(t)$.

Consider a task $T_i$ from the system. The response time distribution $W_{i,j}(t)$ is computed by conditioning on whether or not a backlog of work from equal or higher priority tasks exists when $J_{i,j}$ is released. If no backlog exists, a level-$\phi_i$ busy interval starts at the release of $J_{i,j}$ (which we relabel $J_{i,1}$) and

$$W_{i,1}(t) = \mathcal{P}[w_{i,1}(t) \leq t] \ . \tag{1}$$

Otherwise, the response time distributions for the remaining jobs of $T_i$ in the busy interval are computed in order of their release by

$$W_{i,j}(t) = \mathcal{P}[w_{i,j}(t) \leq t \mid w_{i,j-1}(r_{i,j}) > r_{i,j}] \ . \tag{2}$$

For the highest priority task, the response time distribution of the first job in a busy interval is the same as its execution time distribution. The response time distribution of the subsequent job in the busy interval is computed by convolving the execution time distribution of the task with the distribution of the backlog obtained by conditioning. This process continues until the end of the busy interval. Equations 1 and 2 are also used to compute the response time distributions of the remaining tasks in the system.

We now compute $W_{i,j}(t)$ for $j > 1$. Clearly jobs with a priority higher than $\phi_i$ can execute in the interval $[r_{i,j}, c_{i,j})$. Jobs among $J_{i,1}, J_{i,2}, \ldots, J_{i,j-1}$ that complete after $r_{i,j}$ also execute in this interval. Their effect is taken into account in the conditioning process. To compute $W_{i,j}(\mathrm{t})$, we must still take into account the time demand of jobs of higher priority tasks released in the interval $[r_{i,j}, c_{i,j})$. This is done by dividing $[r_{i,j}, c_{i,j})$ into sub-intervals separated by releases of higher priority jobs and conditioning on whether a backlog of work exists at the start of each sub-interval. For example, suppose that only one higher priority job $J_{k,l}$ is released in the interval $[r_{i,j}, c_{i,j})$ dividing the interval into two sub-intervals, $[r_{i,j}, r_{k,l})$ and $[r_{k,l}, c_{i,j})$. The probability that $J_{i,j}$ will complete by time $t$ before $r_{k,l}$ is

$$W_{i,j}(t) = \mathcal{P}[w_{i,j}(t) \leq t \mid w_{i,j-1}(r_{i,j}) > r_{i,j}] \ , \tag{3}$$

i.e., for $t$ in the first sub-interval $[r_{i,j}, r_{k,l})$, and is

$$\begin{aligned} W_{i,j}(t) = \mathcal{P}[w_{i,j}(t) \leq t \mid w_{i,j-1}(r_{i,j}) > r_{i,j}, w_{i,j}(r_{k,l}) > r_{k,l}] \\ \mathcal{P}[w_{i,j}(r_{k,l}) > r_{k,l}] \ , \end{aligned} \tag{4}$$

for $t$ in the second sub-interval $[r_{k,l}, r_{i,j+1})$. The probability that a job will complete by its deadline is determined by computing $W_{i,j}(D_i)$. Alternatively, the sub-interval distributions can be combined before $W_{i,j}(D_i)$ is computed.

Equations 1 and 2 allow the response time distributions of jobs in a level-$\phi_i$ busy interval to be computed for any combination of initial release times $\{r_{i,1} \mid 1 \leq i < n\}$. In order to compute a lower bound on the probability that jobs complete by their deadlines, the worst-case combination of release times needs to be identified. As discussed previously, an upper bound on the response time of jobs from $T_i$ according to the deterministic TDA is obtained by computing the response times of jobs executed in an in-phase level-$\phi_i$ busy interval. Sadly, we note that it is not longer sufficient for us to consider an in-phase busy interval. The proof that no backlog exists at the instant a job is released simultaneously with the release of jobs of higher priority tasks requires that the maximum total utilization of the system is no greater than one, which is the assumption of deterministic TDA. STDA requires only that the average total utilization of the system is less than one hence some systems may not meet the condition. It is not clear what relationship between the release times of the first jobs in a level-$\phi_i$ busy interval causes some job in $T_i$ to have the maximum possible response time and hence the smallest probability of completing in time. For now, we assume that the first jobs in all tasks are released in-phase and discuss the rationale for this assumption later.

We now turn our attention to the matter of determining when a busy interval ends. We note that since there is a single task per priority level, a level-$\phi_i$ busy interval ends if some job $J_{i,j}$ in $T_i$ completes before the next job $J_{i,j+1}$ is released. Thus we know that the busy interval has surely ended if, for some $j$, $\mathcal{P}[w_{i,j}(r_{i,j+1}) \leq r_{i,j+1}] = 1.0$.[3]

As an example, we now use STDA to analyze the behavior of a system of two tasks shown in Table 2. The execution time of each task is uniformly distributed (with parameters chosen to accentuate the potential for missed deadlines). The worst-case utilization of the system is 1.41 and the mean utilization of the system is 0.71. Consequently, we would expect that some jobs will miss their deadlines. To determine the probability of jobs in each of the tasks missing their deadlines, we apply the procedure outlined above. Because its maximum utilization is less than 1.0, we know that $T_1$ will not miss any deadlines. Therefore we begin the analysis with $T_2$.

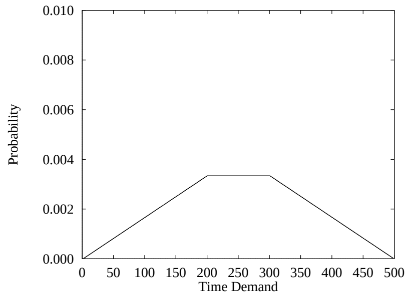
It is apparent that the maximum time demand of $T_2$ in the interval $[0, 400)$ exceeds the time supply because the sum of the maximum utilizations of the two tasks exceeds one. Because $J_{2,1}$ may not have complete by the time $J_{2,2}$ is released, the response time of $J_{2,2}$ may be greater than that of $J_{2,1}$. At the very least we need to compute the response time distributions for $J_{2,1}$ and $J_{2,2}$. To compute the probability that $J_{2,1}$ completes by its deadline, the interval

[3] When multiple tasks have the same priority, jobs from the same priority level must have their response time distributions computed in order of increasing release times. The busy interval will have ended if all jobs with equal or higher priority released before time $t$ have completed by $t$ with probability 1.0.

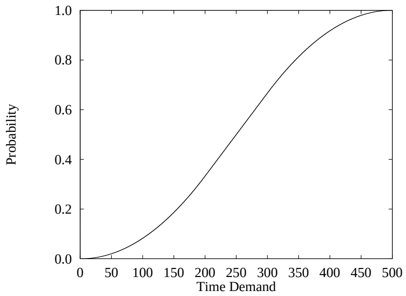
**Table 2.** Parameters of the Tasks in Example #2.

| $T_i$ | $P_i$ | $D_i$ | $E_i^-$ | $\bar{E}_i$ | $E_i^+$ | $U_i^-$ | $\bar{U}_i$ | $U_i^+$ |
|---|---|---|---|---|---|---|---|---|
| $T_1$ | 300 | 300 | 1 | 100 | 199 | 0.0033 | 0.333 | 0.663 |
| $T_2$ | 400 | 400 | 1 | 150 | 299 | 0.0025 | 0.375 | 0.748 |
| Total | | | | | | 0.0058 | 0.708 | 1.411 |

$[0, 400)$ is divided into sub-intervals $[0, 300)$ and $[300, 400)$ due to the release of $J_{1,2}$ at 300. In the first interval, the time demand includes only the execution times of $J_{1,1}$ and $J_{2,1}$. The time demand of the second interval includes the execution time of $J_{1,2}$, as well as the work remaining from the first interval. The probability that a particular time demand occurs is conditioned on whether or not $J_{2,1}$ completes before $J_{1,2}$ is released. We first consider the interval $[0, 300)$. The probability that $J_{2,1}$ will finish by 300 is $\mathcal{P}[w_{2,1}(300) \le 300]$, where $w_{2,1}(t)$ for $0 \le t \le 300$ is computed via the sum $E_1 + E_2$ and has the density function and distribution shown in Fig. 4. The probability that $J_{2,1}$ completes by 300 is 0.668.



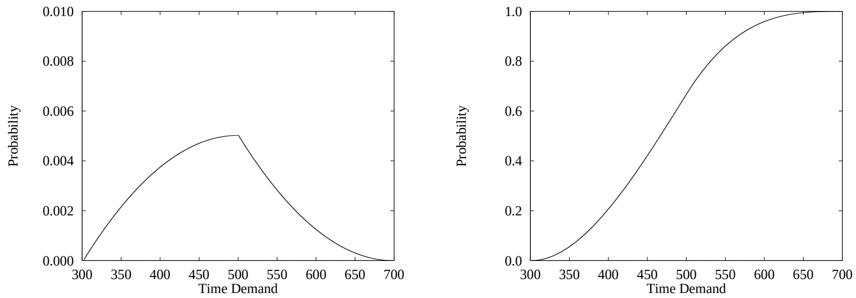

(a) Density




(b) Distribution

**Fig. 4.** Time demand of $J_{2,1}$ over interval $[0, 300)$.

We now compute $\mathcal{P}[w_{2,1}(400) \le 400 \mid w_{2,1}(300) > 300]$ for $t$ in the interval $[300, 400)$. Because $J_{2,1}$ may not have completed by time 300, there are between 0 and 198 time units of work remaining when $J_{1,2}$ is released. The density function for the backlog is the density function of Fig. 4(a) in the range 300–498, normalized to 1.0 as is implied by statistical conditioning. The random variable for the backlog is then added to the execution time of $J_{1,2}$. The resulting density and distribution are given in Fig. 5. The probability that $J_{2,1}$ completes by 400, given that it did not complete by 300, is 0.209 as shown in Fig. 5(b).




(a) Density

(b) Distribution

**Fig. 5.** Time demand of $J_{2,1}$ over interval $[300, 400)$.

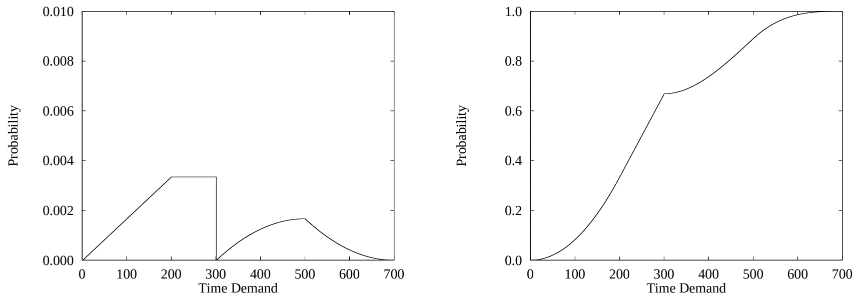
Combining the results of analyzing the two sub-intervals gives us the distribution of the response time of $J_{2,1}$ and thus the probability that $J_{2,1}$ completes by 400 and meets its deadline

$$\mathcal{P}[w_{2,1}(400) \leq 400] = (0.668) + (0.209)(0.332) = 0.738\,. \tag{5}$$

The complete density and distribution functions of the response time of $J_{2,1}$ over the interval $[0, 400)$ are given in Fig. 6. We note that the probability that $J_{2,1}$ will not complete before $r_{2,2}$ is 0.262 so it is also necessary to compute the probability that $J_{2,2}$ completes by its deadline. The analysis proceeds following the same pattern until the busy interval ends. The probability that $J_{2,2}$ completes by its deadline at 800 is 0.994. The probability that $J_{2,3}$ completes by its deadline at 1200 is 1.000. Thus a lower bound on the probability that jobs in $T_2$ meet their deadlines is 0.738.

We now return to the choice of initial phases for tasks. While we do not know what phasing causes a critical instance to occur, we hypothesize that the event occurs so infrequently that the average completion rate is not significantly affected. To test this hypothesis, we performed a series of simulation experiments on a number of systems. For each system, we determine the behavior of the system when each task $T_i$ has a randomly distributed phase in the range $(-P_i, P_i)$ and when all tasks have equal phases, i.e., are released at time 0. (We call a unique combination of phases and actual execution times of the tasks a *run*.) A large number of jobs in each task are released in each run. For each run, a histogram of the response time of the jobs in each task is computed. The histograms of all the runs are averaged, bin by bin, to obtain a histogram representing the average behavior of the tasks of the system. The histograms for in-phase and random-phase releases are then compared.

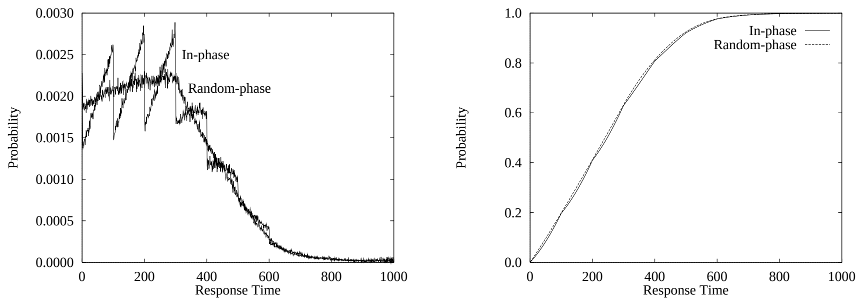
For the tasks in Example 2, we performed 1000 runs for both in-phase and random-phase releases, each run containing the release of at least 1000 jobs in each task. The width of the 95% confidence interval on the profile of the




(a) Density (b) Distribution

**Fig. 6.** Time demand of $J_{2,1}$ over interval $[0, 400)$.

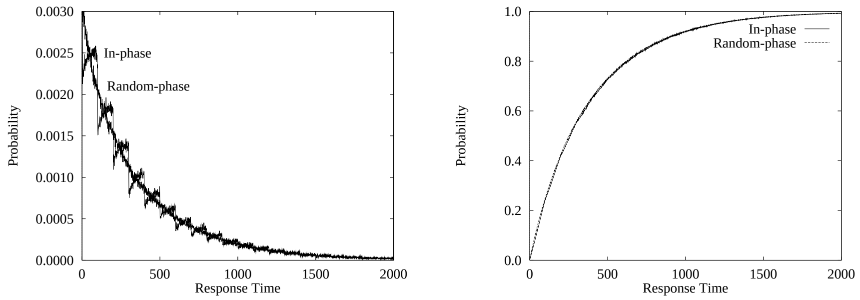
histogram was $\pm 5\%$ or less except in the tail of the density function where the probability was small to begin with. Figure 7 shows the histograms for task $T_2$ from our example.




(a) Density (b) Distribution

**Fig. 7.** Average response times of $T_2$.

As Fig. 7(b) shows, the average response time distribution for in-phase releases bounds the distribution for random-phase releases from below. The average response time density function, Figure 7(a), exhibits a curious saw-tooth behavior for in-phase releases. The behavior is caused by the fixed relationship between the release times of $T_1$ and $T_2$. This relationship causes the completion of jobs in $T_2$ to be delay by jobs in $T_1$ in a periodic manner. The linearly rising shape of each tooth is due to the uniform distribution of the execution time of $T_1$ while the general shape of the curve results from combined effect of the execution time distributions of both $T_1$ and $T_2$. Figure 8 compares the histograms for tasks with the same parameters as our example but with exponential distribution times. Once again, the distribution with in-phase release bounds the distribution with random-phase release from below. Also, the in-phase release curve exhibits a similar saw-tooth shape. However, each tooth has a more rounded shape due to the exponential distribution of $T_1$. Finally, the asymptotically decreasing shape of the density curves indicates the combined effect of the execution time distributions of both tasks.




(a) Density

(b) Distribution

**Fig. 8.** Average response times of $T_2$ (Exponential).

Despite the large number of systems simulated, we have not observed a case where tasks that are released with arbitrary phases have a lower average completion rate than the same tasks that are released in-phase. We therefore use in-phase busy intervals in computing a lower bound on the average completion rate using STDA.

We now compare the lower bound on the probability of meeting deadlines obtained via STDA with the percentage of deadlines met for each task in Table 2. The percentage of the jobs in each task meeting their deadlines was obtained by simulating the behavior of the system for 1000 runs. Each run released and executed 1333 jobs of $T_1$ and 1000 jobs of $T_2$ to produce a response time distribution (in the form of a histogram) for the tasks of the system. Once again, the response time distributions of the runs were averaged, bin by bin, to obtain average response time distributions for the tasks, as well as to assess statistical significance. The behavior of the system was observed when both tasks have identical phases, as well as when the phase of each task $T_i$ is uniformly distributed in the range $(-P_i, P_i)$. As previously observed, the average completion rate for systems in which the tasks are in-phase was lower than the average completion rate for systems in which the tasks have random phases by a small but statistically significant amount at a 95% confidence level. (The simulation results shown below are for the case where the tasks are released in-phase.)

**Table 3.** A comparison of STDA bound with simulation results.

| $T_i$ | STDA | Simulation In-phase | Random-phase | Ratio |
|---|---|---|---|---|
| $T_1$ | 100.0 | $100.0 \pm 0.0$ | $100.0 \pm 0.0$ | 1.000 |
| $T_2$ | 73.8 | $80.8 \pm 0.1$ | $81.2 \pm 0.1$ | 0.913 |

According to Table 3 the probability that jobs complete by their deadlines, as computed by STDA, bounds the percentage of deadlines met from below. The bound differs from the simulation results for $T_2$ by only 8.7%. The difference occurs because STDA computes the worst-case probability that jobs in the first busy interval meet their deadlines rather than the percentage of all jobs in the task that meet their deadlines. In this simple example, simulating the behavior of the two tasks is reasonable. However, for realistic systems with many tasks, simulation requires significantly greater effort than STDA. Hence STDA provides a faster way to determine if the probability of a missed deadline is acceptable.

## 4 Implementing STDA

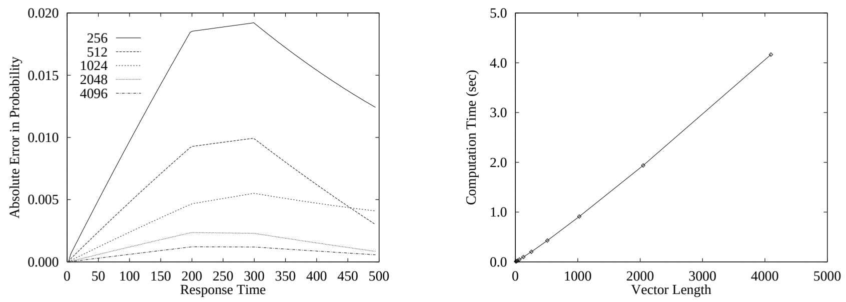
In this section, we discuss an implementation of STDA in the PERTS real-time prototyping environment [6, 7]. PERTS is a tool which facilitates the design and analysis of real-time systems by applying theoretical results, where possible, or by simulating the system to determine its behavior. The issues we discuss are not particular to PERTS and must be addressed by any implementation of STDA.

One of the main operations in STDA is the summing of random variables representing execution times. It is well known that the probability density function of the sum of two statistically independent random variables can be obtained by convolution $f(t) = g(t) \otimes h(t)$.The direct way to perform convolution on a digital computer is to discretize the integral using a constant spacing between samples $f_i = \sum_{j=0}^{N-1} g_i h_{i-j}$. Computing $f$ by direct convolution is an $\mathcal{O}(N^2)$ operation, where $N$ is the number of points in the discrete representations of $g$ and $h$. It has long been known that the asymptotic cost of convolution can be reduced by applying the *Convolution Theorem* $g(t) \otimes h(t) \Longleftrightarrow G(f)H(f)$, where $G(f)$ and $H(f)$ are the Fourier transforms of $g(t)$ and $h(t)$ respectively. The result is an $\mathcal{O}(N \log_2 N)$ algorithm for convolution. There are many descriptions and implementations of the FFT readily available (e.g., [2, 8, 9]).

Three issues need to be considered when using FFT to perform convolution. First, the discrete representations of the probability density functions being convolved must have the same sampling rate and consist of the same number of points. In our application, the vectors containing the discretized probability density functions will almost always have different sample rates and numbers of points as a result of the conditioning process. Thus new vectors must be created by interpolation before every convolution. Since interpolation can be performed in $\mathcal{O}(N \log_2 N)$ time, the asymptotic complexity of convolution is not increased.

Second, sufficient "zero padding" is required to ensure that aliasing does not occur [9]. The length of the vectors are also required to be a power of two. As a result, the vectors are likely to be large and sparsely populated in our application. Our experience indicates that the vectors are often only 50–75% filled with non-zero data. The final issue concerns the number of points used to represent the probability density functions for sufficient accuracy.

Figure 9(a) shows the error between the computed and exact distributions of response time corresponding to Fig. 4(b) as a function of the number of points in the discrete representation. Figure 9(b) shows the computation time as a function of the number of points. In order to maintain acceptable interactive response, we have chosen a default of 1024 points in the PERTS implementation of STDA, which yields a maximum absolute error of slightly over 0.005 for this example.




(a) Accuracy (b) Time

**Fig. 9.** Convolution via FFT versus number of points.

## 5 Conclusions and Future Work

Using hard real-time analysis techniques to design soft real-time system can lead to low resource utilization, increased cost, and poor average performance. In this paper, we have presented the Stochastic Time Demand Analysis method for computing a lower bound on the percentage of jobs in a task that meet their deadlines under a fixed priority scheduling policy. The method enables missed deadlines to be balanced against other design goals such as processor utilization or cost.

In addition to describing the STDA method, we have also performed a simulation study to check the tightness of the bound. For the system used as an example, the bound has less than 10% error. While simulation of the example system of two tasks may not be much more complicated and time consuming than

STDA, the effort to bound the probability of missed deadlines is significantly less than required to simulate systems with many tasks. Hence STDA gives a faster way to determine whether the probability of missed deadlines is acceptable. We have implemented the STDA method in the PERTS environment.

While STDA improves our ability to predict the behavior of soft real-time systems, it is restricted to fixed priority assignments. Similar techniques need to be developed for systems with dynamic priority assignments, such as those scheduled Earliest-Deadline-First. The probability that consecutive jobs will miss their deadlines also needs to be computed, as many soft real-time applications cannot afford to miss more than a certain number of deadlines in a row. Finally, the behavior of systems in which execution times are dependent, periods of jobs vary, jobs share resources, or jobs have precedence constraints between them needs to be considered.

## References


[1] A. K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. Technical Report BUCS-TR-98-010, Boston University, 1998.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

[3] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th IEEE Real-Time Systems Symposium*, pages 201–209, December 1990.

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

[5] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.

[6] J. W. S. Liu, C. L. Liu, Z. Deng, T. S. Tia, J. Sun, M. Storch, D. Hull, J. L. Redondo, R. Bettati, and A. Silberman. PERTS: A prototyping environment for real-time systems. *International Journal of Software Engineering and Knowledge Engineering*, 6(2):161–177, 1996.

[7] J. W. S. Liu, J. L. Redondo, Z. Deng, T. S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. K. Shih. PERTS: A prototyping environment for real-time systems. In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, pages 184–188, Raleigh-Durham, North Carolina, December 1993.

[8] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms.* Springer-Verlag, second edition, 1982.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[10] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings, Real-Time Technology and Applications Symposium*, pages 164–173, Chicago, Illinois, May 1995. IEEE.