Some Notes on Stochastic Analysis Using EDF Scheduling

José Luis Díaz, José María López
Universidad de Oviedo

Technical report SNSAUES07
September 2007

Abstract

This technical report describes how to apply the stochastic analysis of real-time systems presented in [3] to EDF systems. Although the stochastic analysis is general enough to cover both fixed-priority and EDF scheduling, and a detailed explanation on how to deal with EDF is published also in [3] (section 4.4), the lack of published examples using EDF make it difficult to understand.

In addition, we present in this technical report a new approach to apply the stochastic analysis to EDF, which is mathematically equivalent to the one presented in [3], but it is conceptually much simpler. Then, we refine this approach in order to allow a more efficient implementation in an analysis tool.

1 Introduction

Real-time systems must provide some kind of guarantee about the ratio of deadline misses. Classical analysis of hard real-time systems focused on guaranteeing that all deadlines (100%) are met. In order to provide such a guarantee, the worst-case scenario has to be found, and the worst-case response time in that scenario has to be computed. If this worst-case response time is not greater than the deadline for all tasks in the system, the system is feasible. Otherwise it is deemed unfeasible. This over-pessimistic approach leads to oversized systems, in order to provide such a guarantee.

Stochastic analysis of real-time systems focuses on computing the probability of deadline misses. This does not require any worst-case scenario, nor the use of a single-valued worst-case computation time. Instead, all possible computation times and their probabilities are required, and all possible scenarios are analyzed. The result of the analysis is the probability function (PF) of the response times of all tasks in the system. This probability function gives the probability of each possible response time. From it, the probability of deadline misses is easily derived.

The problem of obtaining the PF of the response time, given the periods, offsets, priorities and the PFs of the computation times of the tasks, is apparently unaffordable. This is why some authors imposed simplifications in the model [8, 5, 6] or in the scheduler [2, 1].

The first analysis which did not impose any of these restriction was published in [3] and [7], and then expanded in [4]. The proposed analysis is general enough to be applicable both to fixed-priority and EDF schedulers. However, the explanations about how to apply the analysis to the EDF case in [3, 7] are unnecessarily complex. In addition, the published examples, which clarify how the analysis is carried out, are fixed-priority systems in all cases. As a consequence, the algorithm for applying the proposed analysis to the EDF case remains difficult to understand.

In this report we present a new approach to EDF, conceptually simpler than the one published before. In addition we provide a detailed step-by-step example.

2 System model and overview of the analysis

In this section we formalize the system model used in the analysis, and summarize the algorithm.

2.1 System model

The system is composed of a set of N independent periodic tasks $S = \{\tau_1, \ldots, \tau_i, \ldots, \tau_N\}$, each task τ_i being defined by the tuple $(T_i, \Phi_i, \mathcal{C}_i, D_i, M_i)$, where T_i is the period of the task, Φ_i its initial phase, $\mathcal{C}_i$ its execution time and the pair (D_i, M_i) define the real-time constraint of the task.

The execution time is a discrete random variable[1] with a known probability function (PF), denoted by $f_{\mathcal{C}_i}(\cdot)$, where $f_{\mathcal{C}_i}(c) = \mathbb{P}\{\mathcal{C}_i{=}c\}$. Alternatively, the execution time distribution can also be specified using its cumulative distribution function (CDF), denoted by $F_{\mathcal{C}_i}(\cdot)$, where $F_{\mathcal{C}_i}(x) = \sum_{c=0}^{x} f_{\mathcal{C}_i}(c)$. In the stochastic analysis three system utilizations are defined, namely $U^{\min}$, $U^{\max}$ and $\bar{U}$, which are calculated using the minimum, maximum and average task execution times, respectively.

[1] Throughout this report we use a calligraphic typeface to denote random variables, e.g. $\mathcal{C}$, $\mathcal{W}$, $\mathcal{R}$, etc.

Each periodic task gives rise to an infinite sequence of jobs, Γ_j, with deterministic release times λ_j. Each job requires an execution time which is a random variable whose distribution is given by the probability function of the task it comes from, $f_{\mathcal{C}_i}(\cdot)$, and it is assumed to be independent of other jobs of the same task and those of other tasks. The response time of a job Γ_j is a random variable, $\mathcal{R}_j$, whose probability function has to be obtained by the analysis. D_i is the task relative deadline and M_i the maximum allowable probability of missing it. Task τ_i is said to be schedulable if $\mathbb{P}\{\mathcal{R}_i > D_i\} \leq M_i$, $\mathcal{R}_i$ being the response time of τ_i.

The scheduling policy we assume is a general, preemptive, priority-driven policy that assigns a static priority to each job and schedules jobs according to this priority. The scheduler guarantees that the running job is the one with the highest priority among the ready jobs. We are not concerned with the policy used to assign priorities to jobs, as long as they are assigned in a deterministic way. This model includes well known fixed priority policies such as *Deadline Monotonic* (DM), and non-fixed priority policies such as *Earliest Deadline First* (EDF).

2.2 Analysis

The response time of a job Γ_j is given by

$$\mathcal{R}_j = \mathcal{W}(\lambda_j) + \mathcal{C}_j + \mathcal{I}_j \tag{1}$$

where $\mathcal{W}(\lambda_j)$ is the backlog at time λ_j, which represents the workload of priorities P_j and higher that have not yet been processed immediately prior to λ_j, the release time of Γ_j. Term $\mathcal{C}_j$ is the execution time of job Γ_j and $\mathcal{I}_j$ is the interference in Γ_j of all the jobs of higher priority than job Γ_j released at or after job Γ_j. Note that all the terms in Equation (1) are random variables. This equation is the stochastic counterpart of a well known deterministic equation, which provides the response time of a job under any preemptive priority-driven scheduling policy.

In order to simplify the mathematical expressions, all the jobs with priority lower than Γ_j will henceforward be removed, since they do not affect its response time.

The backlog at the release time of any job Γ_j, denoted $\mathcal{W}(\lambda_j)$, can be calculated using the following iterative equation:

$$\begin{aligned} &\mathcal{W}(\lambda_1) = 0 \\ &\mathcal{W}(\lambda_k) = \text{SHRINK}(\mathcal{W}(\lambda_{k-1}) + \mathcal{C}_{k-1}, \lambda_k - \lambda_{k-1}) \quad \text{for } k > 1 \end{aligned} \tag{2}$$

where λ_1 is the release time of the first high priority job. Note that the backlog at λ_j does not account for the computation time of job Γ_j. Equation (2) starts with zero backlog and continues with a new iteration for each high priority job released before Γ_j. Each iteration step requires the following operations on the Probability Functions (PFs):

- Convolution of the PF of the backlog calculated in the previous iteration, at time λ_{k-1}, with the PF of the execution time $\mathcal{C}_{k-1}$, of the high priority job released at that time.
- Shift $\Delta = (\lambda_k - \lambda_{k-1})$ time units to the left of the result of the previous step.
- Accumulate at zero the probabilities of negative time values.

The last two operations are summarized in function SHRINK$(\cdot)$, whose PF is defined by Equation (3).

$$f_{\text{SHRINK}(\mathcal{W},\Delta)}(x) = \begin{cases} 0 & \text{if } x < 0 \\ \sum_{w=-\infty}^{0} f_{\mathcal{W}}(w+\Delta) & \text{if } x = 0 \\ f_{\mathcal{W}}(x+\Delta) & \text{if } x > 0 \end{cases} \tag{3}$$

Once the backlog at the release time of Γ_j is known, it is added to its execution time $\mathcal{C}_j$, as given by Equation (1). The PF of the addition is calculated by convolving the PF of both random variables. This provides the *partial response time* $\mathcal{R}_j^{[0,\lambda_{j+1}-\lambda_j]} = \mathcal{W}(\lambda_j) + \mathcal{C}_j$, where λ_{j+1} is the release time of the first high priority job released at or after Γ_j. Partial response time $\mathcal{R}_j^{[0,\lambda_{j+1}-\lambda_j]}$ is the response time of job Γ_j assuming that high priority jobs Γ_{j+1} and subsequent do no exist, so they can not interfere with Γ_j. Therefore, $\mathbb{P}\{\mathcal{R}_j = r\} = \mathbb{P}\{\mathcal{R}_j^{[0,\lambda_{j+1}-\lambda_j]} = r\}$ for all $r \in [0, \lambda_{j+1} - \lambda_j]$.

The sequence of partial response times of Γ_j can be calculated using Equation (4)

$$\begin{aligned} \mathcal{R}_j^{[0,\lambda_{j+1}-\lambda_j]} &= \mathcal{W}(\lambda_j) + \mathcal{C}_j \\ \mathcal{R}_j^{[0,\lambda_{k+1}-\lambda_j]} &= \text{AF}(\mathcal{R}_j^{[0,\lambda_k-\lambda_j]}, \lambda_k - \lambda_j, \mathcal{C}_k) \quad \text{for } k > j \end{aligned} \tag{4}$$

Equation (4) is an iterative equation, which starts with partial response time $\mathcal{R}_j^{[0,\lambda_{j+1}-\lambda_j]}$ and continues with a new iteration for each high priority job released at or after Γ_j. Each iteration step requires a special addition of the previous partial response time, $\mathcal{R}_j^{[0,\lambda_k-\lambda_j]}$, with the execution time of the next high priority job, $\mathcal{C}_k$. The special "addition", called *Addition From* (AF), provides a random variable whose PF is defined by Equation (5)

$$f_{\text{AF}(\mathcal{R},\Delta,\mathcal{C})}(x) = \begin{cases} f_{\mathcal{R}}(x) & \text{for } x \leq \Delta \\ \sum_{i=\Delta+1}^{\infty} f_{\mathcal{R}}(i) \cdot f_{\mathcal{C}}(x-i) & \text{for } x > \Delta \end{cases} \tag{5}$$

This PF is calculated by performing a special convolve operation, named *convolve-from*. It is special in the sense that it leaves the PF of $\mathcal{R}$ in interval $[0,\Delta]$ unchanged, convolving the PF of $\mathcal{R}$ with that of $\mathcal{C}$ only in interval $[\Delta+1,\infty]$.

Note that under EDF this iterative procedure can stop when all future jobs have absolute deadlines greater than $(\lambda_j + D_j)$, because all these jobs are lower priority jobs, and so cannot interfere with the job under analysis. At maximum, this condition will be met for jobs arriving after the instant $(\lambda_j + D_j)$. Due to this fact, this step will usually require few iterations for EDF systems.

On the contrary, under fixed-priority scheduling there is an infinite number of interfering jobs (all future jobs from higher priority tasks). This requires an infinite number of iterations to obtain the exact distribution of the response time of the job. However, in order to calculate the probability of Γ_j missing its deadline D_j, it is enough to iterate on Equation (4) until $\delta = \lambda_{k+1} - \lambda_j \geq D_j$, because $\mathbb{P}\{\mathcal{R}_j > D_j\} = 1 - \mathbb{P}\{\mathcal{R}_j \geq D_j\} = 1 - \sum_{k=0}^{k=D_j} \mathbb{P}\{\mathcal{R}_j^{[0,\delta]} = k\}$.

In theory, the probability of a task missing its deadline is calculated by averaging the probabilities of all its jobs missing that deadline, but in practice the number of these jobs is infinite. However, when $\bar{U} < 1$ the system reaches a periodic steady-state [3]. In the

	Task τ_1	Task τ_2
Phase	$\Phi_1 = 20$	$\Phi_2 = 50$
Period	$T_1 = 40$	$T_2 = 60$
Deadline	$D_1 = 50$	$D_2 = 90$
Execution Time	$\mathcal{C}_1 = \mathcal{C}_2 = \mathcal{C}$	

Table 1: Parameters of the example system

steady-state, the probability of a job missing its deadline becomes constant for the same job released one, two or any number of hyperperiods later. Thus, the probability of a task missing its deadline can be calculated by considering only those of its jobs released in one hyperperiod.

The procedure above outlined makes no assumption about the scheduler, as long as the priorities of the jobs do not change during job execution. Thus, it is valid both for fixed-priority and EDF schedulers. However, some observations should be made:

- The term $\mathcal{W}(\lambda_j)$ in eq. (1) represents the backlog to be completed before job Γ_j can start its own execution. For the steady-state, this term is computed differently under fixed-priority and EDF schedulers.

  In the case of fixed-priorities, $\mathcal{W}(\lambda_j)$ is the sum of all execution times, still not serviced, of all jobs belonging to tasks with priority greater than (or equal to) P_j.

  In the case of EDF, $\mathcal{W}(\lambda_j)$ is the sum of all the execution times, still not serviced, of all preceding jobs which have earlier deadline than job Γ_j, no matter to which task they belong. These are the jobs whose absolute deadline is less than, or equal to, $(\lambda_j + D_j)$.

- Once $\mathcal{W}(\lambda_j)$ is obtained, the rest of the analysis is carried out in the same way for fixed-priorities and EDF schedulers. However, as noted in previously, under EDF the number of jobs which can cause interference is finite (indeed, it is usually very small), and thus it is possible to obtain the exact response time of each job.

In the next section, a detailed example is solved; afterwards, the procedure used to solve it is generalized.

3 Example for EDF

Table 1 shows the parameters of a real-time system. The execution times of both tasks are identically distributed random variables, following the probability function given in Table 2. The maximum utilization (when all jobs use their WCET) is 2.0833, and the average utilization (when all jobs use their average execution time) is 0.9417. Since the maximum utilization is greater than one, the backlog computation must iterate over several hyperperiods until convergence is reached. Since the average utilization is less than one, the convergence is guaranteed.

Figure 1 shows the release pattern of the jobs across the two first hyperperiods. In this figure, the release instants of the jobs are represented by upwards pointing arrows, while their absolute deadline are represented by downwards pointing hollow arrows. The release instant and the absolute deadline of each job are connected by a horizontal line.

c	$\mathbb{P}\{\mathcal{C}=c\}$
10	0.1
20	0.4
21	0.2
22	0.2
50	0.1

Table 2: Probability function of $\mathcal{C}$

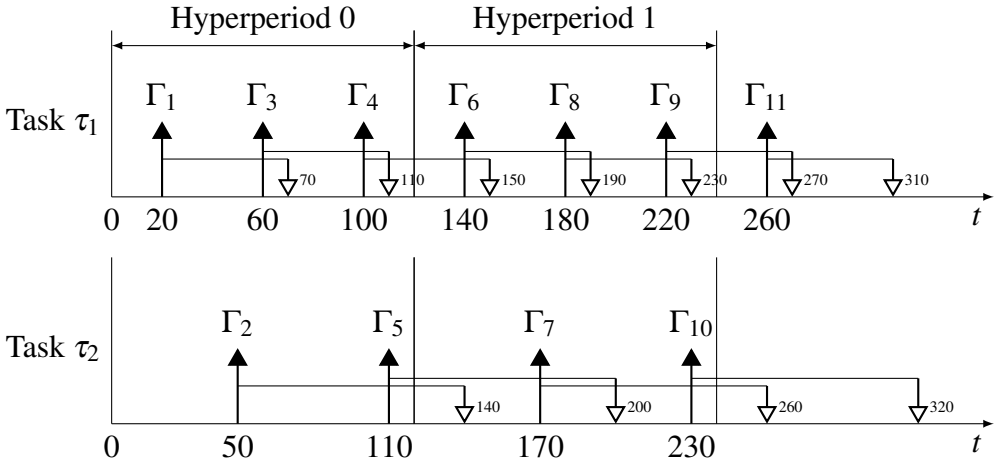

Figure 1: Release pattern over the two first hyperperiods

In this figure, jobs were numbered by arrival ordering, no matter the task to which they belong.

Figure 2 is a generalization of Fig. 1. It can be seen that all hyperperiods repeat the same pattern, and the only difference is that the value of k increases. This notation will be useful for reasoning about the steady-state hyperperiod.

We are interested in the probability function of the response time of both tasks, τ_1 and τ_2. Figure 3 gives a quick overview of the computation procedure. Next, we detail the required steps.

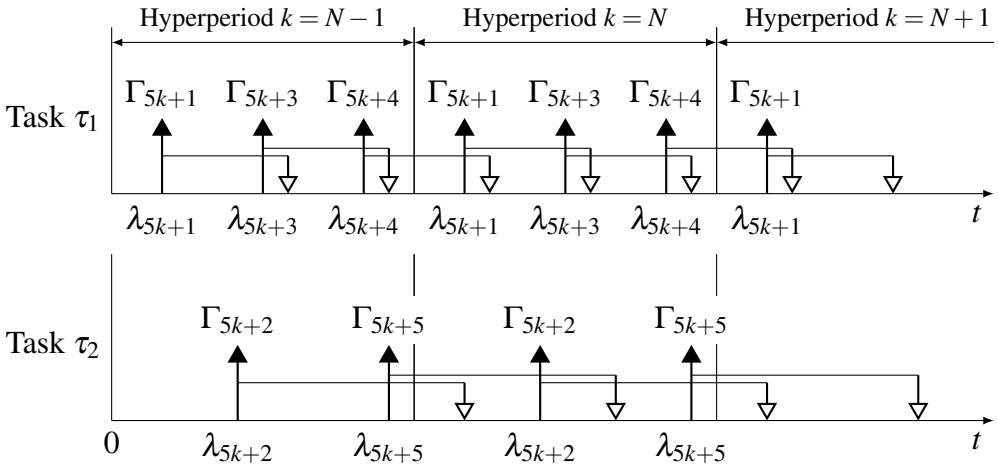

Figure 2: Generalization of Figure 1

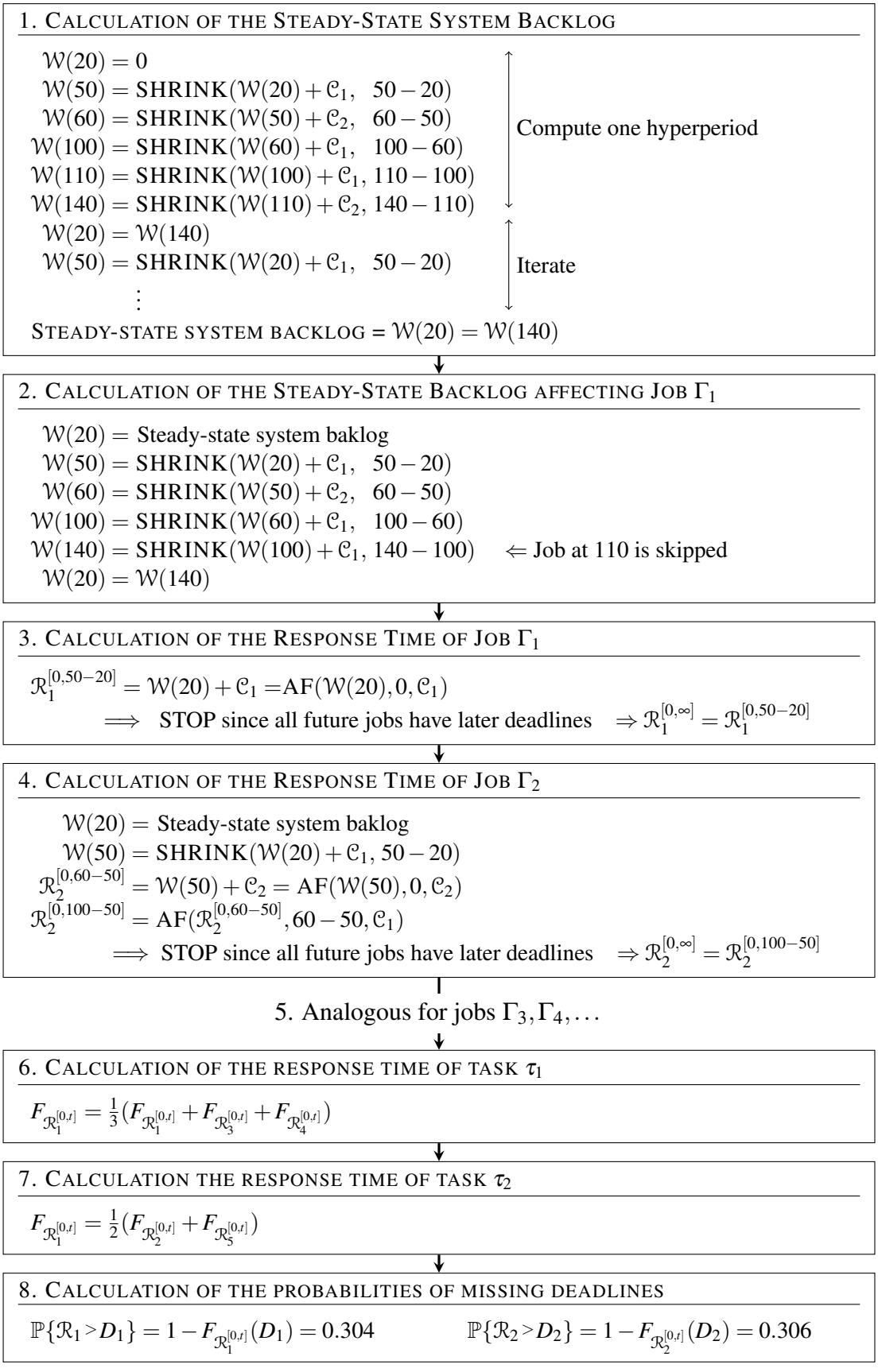

Figure 3: Example of calculation of the random response time of a task

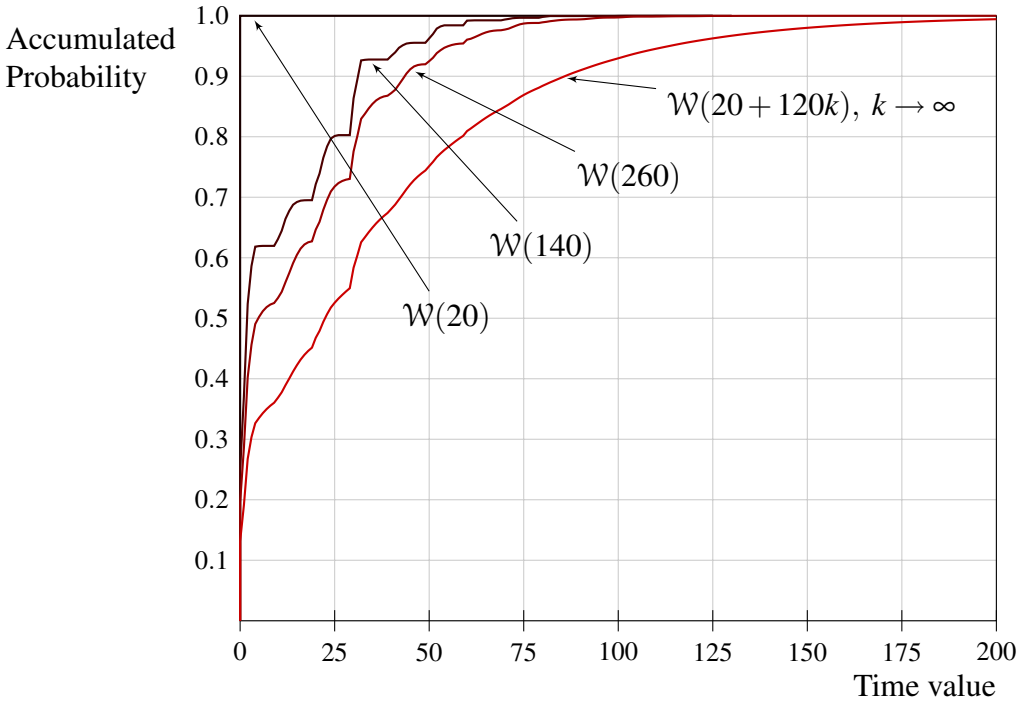

Figure 4: Cumulative distribution functions of system backlog at the release instant of the first job in the hyperperiod.

1. Calculation of the steady-state system backlog The first step in the procedure is to obtain the system backlog at the instant in which the first job in the steady-state hyperperiod is released. In our case, the first job in the hyperperiod belongs to τ_1, and it is Γ_1 in the first hyperperiod, Γ_6 in the second hyperperiod, and so on. In general, it is job Γ_{5k+1}, being k the hyperperiod under consideration, numbered from zero (refer to Fig. 2). This job is released at instants 20, 140, 260,...; in general at $\lambda_{5k+1} = 120k+20$. Therefore, we need to obtain the system backlog at instant $(120k+20)$, when $k \to \infty$.

The procedure for obtaining this backlog consists of iterating Eq. (2) through several hyperperiods, until the same result is obtained in two consecutive hyperperiods. Figure 4 shows how the convergence is reached and the cumulative probability function of this system backlog.

2. Calculation of the steady-state backlog affecting job Γ_1 Let us suppose that convergence is reached for hyperperiod $k = N$ in Fig. 2. The system backlog obtained in step 1 is then the system backlog at instant λ_{5k+1} in this hyperperiod. This system backlog includes the contribution of all preceding jobs. However, looking at Figure 2, it is clear that job Γ_{5k+5} from the hyperperiod $(N-1)$ does not affect to the job under analysis, because its absolute deadline is greater. So, the system backlog obtained in step 1 is not directly usable to compute the response time of Γ_{5k+1}. In general, the adequacy of the computed backlog can be algorithmically assessed if, while computing the steady-state system backlog, a variable is used to keep the maximum of the absolute deadlines of the involved jobs.

The algorithm used to obtain this backlog does not maintain any "memory" of previous steps, so it is not possible to "rewind" the computation up to the point in which the extra job was included, and re-compute the backlog from this point. Instead, it is

much simpler to apply the backlog computation procedure for one more hyperperiod, until reaching Γ_{5k+1} again in the next hyperperiod, $(N+1)$. Since the steady state is assumed to have been reached, the response time of job Γ_{5k+1} does not depend on k. In this additional iteration through hyperperiod N, only jobs with absolute deadline less than $(\lambda_{5k+1}+D_1)$, for $k=(N+1)$, have to be included. In this example this means jobs Γ_{5k+1}, Γ_{5k+2}, Γ_{5k+3} and Γ_{5k+4} from hyperperiod N (see Fig. 2), excluding Γ_{5k+5} from hyperperiod N, whose absolute deadline is greater than that of Γ_{5k+1} from hyperperiod $(N+1)$. A simple way of skipping this job is to consider its computation time equal to zero, while applying Eq. (2).

The rest of jobs in the hyperperiod do not suffer from this problem, so the backlog affecting them can be directly derived from the system backlog at λ_{5k+1}, $\mathcal{W}(20)$.

3. Calculation of the response time of job Γ_1 Once the backlog affecting the job is obtained, the response time of the job is computed using Eq. (4) as explained in section 2.2. Note that this step use the same algorithm for fixed-priority and EDF systems, and the only difference is the stopping condition. Also note that, under EDF, the algorithm requires usually a small number of iterations, because the most habitual case is that future jobs have greater absolute deadline (lower priority) and then they do not cause interference.

In particular, looking at job Γ_{5k+1} in Fig. 2, it can be seen that al future jobs have later deadlines, so they are lower priority jobs which do not interfere job Γ_{5k+1}. In this case, with a single convolution, we have obtained the exact PF of the response time, valid for all t.

4. Calculation of the response time of job Γ_2 The case of job Γ_{5k+2} (which belong to task τ_2) is slightly more interesting. Looking again at Fig. 2, it can be seen that job Γ_{5k+3} (belonging to task τ_1) has an earlier absolute deadline, so it causes interference. This is however the only job which interferes, so in this case the exact PF of response time of Γ_{5k+2} is obtained after one "normal" addition and one "addition from" operation.

5. Analogous for jobs Γ_3, Γ_4, ... The response time PF for the rest of the jobs in the steady-state hyperperiod is computed in the same way. Note also that the computation of the response time requires a single "normal" addition for all jobs in the hyperperiod, except for job Γ_2 (as explained above) and job Γ_5, for the same reasons than job Γ_2. In this case, Γ_5 suffers interference from job Γ_1 released in the next hyperperiod.

6. Calculation of the response time of task τ_1 Once the PF of the response time of each job is found, the PF of the response time each task is obtained by averaging the response time PFs of the jobs belonging to this task. In the case of τ_1, the jobs whose response time has to be averaged are Γ_1, Γ_3, and Γ_4. The result is shown in Figure 5.

7. Calculation of the response time of task τ_2 It is obtained in the same way, by averaging the response time PF of jobs Γ_2 and Γ_5. The result is also shown in Figure 5.

8. Calculation of the probabilities of missing deadlines Once the cumulative probability distributions of the response time of the tasks are obtained, the probability of meeting the deadline can be easily computed, by simply evaluating those functions for

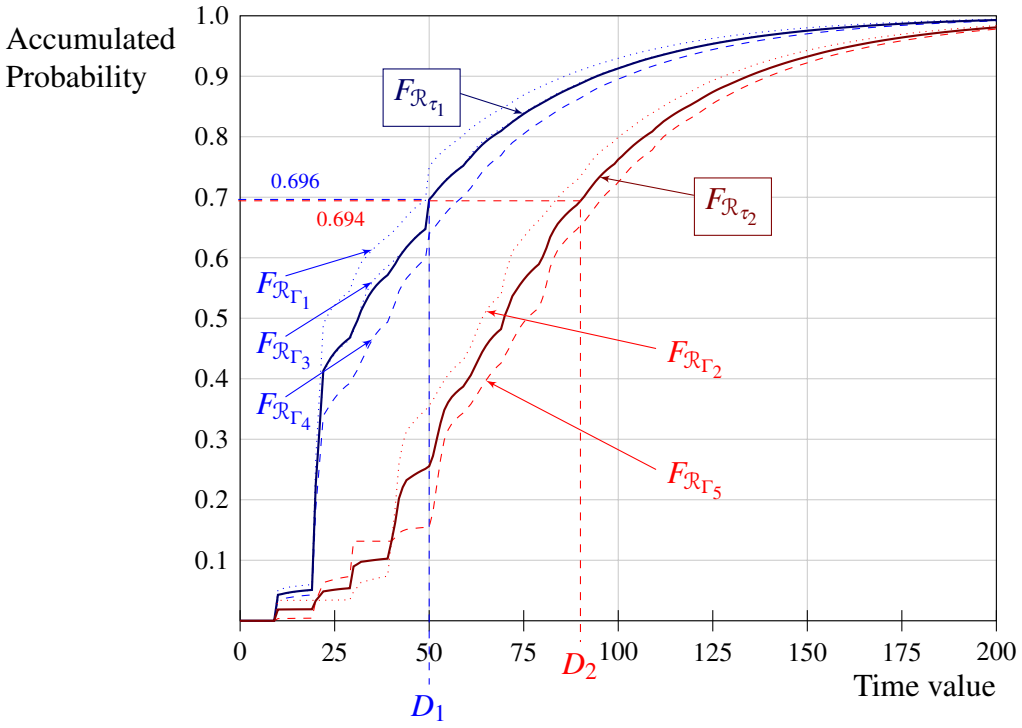

Figure 5: Cumulative distribution functions of response time for both tasks in the system

the deadline. Figure 5 shows this evaluation, and the result for each task. The deadline miss probability is 1 minus the probability of deadline meeting, and the result is $(1-0.696) = 0.304$ for τ_1, and $(1-0.694) = 0.306$ for τ_2.

4 Generalization of the backlog computation method

Firstly, the total system backlog at the instant of activation of the first job in the steady-state hyperperiod is computed by iteratively applying Eq. (2), including all the jobs in the system, no matter their absolute deadlines (i.e., irrespective of their priorities). As described for fixed-priorities, this computation may require iterating over several hyperperiods until convergence is reached. Let $\mathcal{W}$ be the steady-state system backlog ($\mathcal{W}(20$ in the previous example), and let $D_{\mathcal{W}}$ be the maximum of all absolute deadlines for all jobs whose workload was included in $\mathcal{W}$.

Then, for each job, Γ_j, in the hyperperiod the backlog at its release time, $\mathcal{W}(\lambda_j)$, can be computed as follows:

- If $D_{\mathcal{W}}$ is not greater than the absolute deadline of job Γ_j (i.e.: if $D_{\mathcal{W}} \leq \lambda_j + D_j$), the pre-calculated backlog $\mathcal{W}$ is correct, since it includes only jobs with priority greater than job Γ_j. In this case, Eq. (2) is iteratively applied, starting with the steady-state system backlog, until reaching the release time of job Γ_j. While applying this equation, only jobs with absolute deadline not greater than $(\lambda_j + D_j)$ are considered.

- Otherwise, i.e: if $D_{\mathcal{W}} > \lambda_j + D_j$, the pre-calculated backlog is not valid, since it includes at least one job with a priority less than priority of job Γ_j. In this case, we must "discount" the workload caused by these jobs.

In [3] a elaborated method is presented to "rewind" the computation up to the point in which all jobs included have earlier deadlines than the job under analysis. This method is unnecessarily complex, because rather than "rewinding" the algorithm, it is much simpler to advance it up to the next hyperperiod. This is possible because, once the steady-state is reached, all future hyperperiods are equivalent.

The new proposed method consists of actualizing $\mathcal{W}$ by repeatedly applying Eq. (2) until the next hyperperiod is reached, but now including only the jobs with absolute deadline earlier than the absolute deadline of Γ_j in next hyperperiod. This procedure is repeated if necessary over several hyperperiods more, until no jobs with later absolute deadline are included in $\mathcal{W}$. Note that the relative deadlines of the jobs are rarely greater than the length of the hyperperiod, which guarantees that this condition is usually met iterating on a single iteration. Once the condition is met, we are in the first case.

5 Optimization of the backlog computation method

Previous section described a general method to calculate the steady-state backlog affecting a job Γ_j at its release time, i.e, term $\mathcal{W}(\lambda_j)$ in Equation (1). Starting from the steady-state system backlog, the backlog affecting a job was calculated by iterating on Equation (2) until λ_j, skipping all the jobs with absolute deadline higher than the absolute deadline of Γ_j. The only nuisance appeared when the steady-state system backlog were not valid for Γ_j, since it included jobs with absolute deadline higher than that Γ_j. However, the solution was simple, just calculate the backlog for job Γ_j released one or more hyperperiods later.

Although this calculation method is correct, its performance can be improved. There are two simple optimizations that can greatly decrease the computational cost.

1. The steady-state backlog affecting the first job of a task is a valid starting point in the calculation of the steady-state backlog affecting its second job. In the same way, the steady-state backlog affecting the second job of a task is a valid starting point in the calculation of the steady-state backlog affecting its third job, and so on. For example, the steady-state backlog at $t = 110$ affecting job Γ_5 of task τ_2 in the example, can be calculated by iterating from the steady-state backlog at $t = 50$ affecting job Γ_2 of the same task. That is to say, we do not have to iterate from the steady-state system backlog at $t = 20$. The proof is simple. Since both Γ_2 and Γ_5 belong to the same task, the absolute deadline of Γ_2 is lower than that of Γ_5, so the backlog affecting Γ_2 is valid for Γ_5.

2. The second optimization can be applied whe the steady-state system backlog is not valid for a job, as happened with job Γ_1 in the example. In that case, we iterated starting from the steady-state system backlog during one hyperperiod more until reaching the same job one hyperperiod later, skipping all the jobs with higher absolute deadline. However, we can observe in Figure 1 that the steady-state system backlog is valid for Γ_3, since the absolute deadline of Γ_3 is higher than the maximum absolute deadline of the jobs included in the steady-state system backlog. Therefore, the steady-state system backlog is valid to calculate the backlog affecting Γ_3 at its release time. Next, applying the first optimization, the steady-state backlog affecting Γ_3 at its release time is a valid starting point to calculate the

steady-state backlog of Γ_4. Finally, the steady-state backlog affecting Γ_4 at its release time is a valid starting point to calculate the steady-state backlog of Γ_6. Now, in order to calculate the response time of τ_1 we consider jobs Γ_3, Γ_4 and Γ_6, instead of Γ_1, Γ_3 and Γ_4. However, both sets of jobs are equivalent, because the system is supposed to be in the steady-state.

Since stochastic analysis of real-time systems is costly in computational terms, it is suitable to apply the optimization techniques showed in this section when dealing with complex real-time systems.

A Implementation

This appendix provides a minimal implementation of the procedure described in the report. It is written in Python language. The aim is not speed, efficiency or elegance. Instead, the code tries to mimic the steps and notation used in the report, so in some sense this is a repetition of Fig. 3. But it is an *executable* description, so anyone can try it and obtain the same results presented in the report. If you want to generate plots like Figs. 4 and 5, you may insert calls to function `PF_save` at appropiates points in the code. Refer to last lines in the code for the appropiate syntax.

The code in this appendix is available for download at `http://www.atc.uniovi.es/rsa/starts/downloads/python-example.tgz`.

In addition, a much more complete and versatile tool called *stochan* was developed by the authors. This tool is capable of dealing with complex systems including blocking, release jitter, mixed priorities models, different rounding methods, etc. The tool is written in C language, and it is optimized for speed and memory consumption. *Stochan* is available for download (in binary form for Windows and Linux platforms) at `http://www.atc.uniovi.es/rsa/starts/tools.php`

The reader may be interested in knowing the time required to perform the analysis for the system presented in this report. The parameters of the system were carefully choosen in order to have an interesting behaviour with a small number of jobs. The high average system utilization, 0.9417, and specially, the use of computation times greater than periods[2], causes a slow convergence of the iterative method. In order to obtain the stationary system backlog with an error less than 10^{-6}, it was neccesary to iterate over 125 hyperperiods and the resulting backlog probability function contained 11462 points. Using the python code, the time required to complete the analysis was 2 minutes 27 seconds. Using *stochan* it was only 0.915 seconds.

A.1 Python code

Main procedure (file `example-edf.py`)

```
15 from analysis import PF, PDF, shrink, convolve, AF,\
                difference, average, PF_save
17
   # Execution time probabilities (both tasks have the same distribution)
19 C=PF({10: 0.1, 20: 0.4, 21: 0.2, 22: 0.2, 50: 0.1})
```

[2]Assuming that jobs missing their deadlines are removed from the running queue by the operating system, execution time 50 of task τ_1 may be substituted by execution time $D_1 + 1 = 41$, which would reduce the pessimism and decrease the analysis time.

```
################################################################
# CALCULATION OF THE STEADY STATE SYSTEM BACKLOG
################################################################

# Set initial backlog equal to zero with probability 1
W0=PF({ 0:1 })

# Advance to instant 20, in which the first job arrives
W20=shrink(W0,20)

# Iterative procedure
iteration=1; diff=1
while(diff>0.00001):
  W50 = shrink(convolve(C, W20), 50-20)
  W60 = shrink(convolve(C, W50), 60-50)
  W100 = shrink(convolve(C, W60), 100-60)
  W110 = shrink(convolve(C, W100), 110-100)
  W140 = shrink(convolve(C, W110), 140-110)
  diff = difference(W140, W20)
  iteration=iteration+1
  # Copy the backlog at the end of the previous hyperperiod
  # as the backlog at the beginning of the new hyperperiod
  W20=W140
  print "# Iteration %d. diff=%2.10f. Backlog size=%d" %\
       (iteration, diff, len(W140))

################################################################
# CALCULATION OF THE STEADY STATE BACKLOG AFFECTING JOB 1
################################################################
W50 =shrink(convolve(C, W20), 50-20)
W60 =shrink(convolve(C, W50), 60-50)
W100=shrink(convolve(C, W60), 100-60)
W110=shrink(convolve(C, W100), 110-100)
W140=shrink(W110, 140-110) # No convolve in this step
                     # this way Job5 (at 110) is excluded
W20=W140

# CALCULATION OF THE RESPONSE TIME OF Job1
################################################################
Rj1=convolve(W20, C)

# CALCULATION OF THE RESPONSE TIME OF Job2
################################################################
Rj2=convolve(W50, C)
Rj2=AF(Rj2, 60-50, C) # Interference of job3

# CALCULATION OF THE RESPONSE TIME OF Job3
################################################################
# The backlog affecting Job3 must be recomputed, since W60
# includes Job2, with lower priority
W60 =shrink(W50, 60-50) # Do not include Job2
Rj3=convolve(W60, C)

# CALCULATION OF THE RESPONSE TIME OF Job4
################################################################
Rj4=convolve(W100, C)

# CALCULATION OF THE RESPONSE TIME OF Job5
################################################################
Rj5=convolve(W110, C)
Rj5=AF(Rj5, 140-110, C) # Interference of job1 in next hyperperiod

# CALCULATION OF THE RESPONSE TIME OF TASK1
################################################################
Rt1=average([Rj1, Rj3, Rj4])

# CALCULATION OF THE RESPONSE TIME OF TASK2
################################################################
Rt2=average([Rj2, Rj5])

################################################################
# CALCULATION OF PROBABILITIES
################################################################
```

```
D1=50 # Deadlines
D2=90 #
print "Probability of deadline misses for Task1 = %2.10f" %\
     (1-PDF(Rt1)[D1])
print "Probability of deadline misses for Task2 = %2.10f" %\
     (1-PDF(Rt2)[D2])
# Save cumulative distributions for plotting (only first 201 points)
PF_save(PDF(Rt1[:201]), "Task-1-pdf.dat")
PF_save(PDF(Rt2[:201]), "Task-2-pdf.dat")
```

Analysis module (file `analysis.py`)

```
"""
This module implements a minimal set of funtions which allow to perform
the stochastic analysis of real-time systems in a semi-manual way.
For more information see technical report available at
http://www.atc.uniovi.es/research/SNSAUE07.pdf
"""

# CORE SET OF FUNCTIONS
# ============================================================
def convolve(f,g):
  """
  Implements the convolution of two probability functions
  """
  r=[0.0]*(len(f)+len(g)-1) # Make up space for result
  for i in range(len(f)):
    for j in range(len(g)):
      r[i+j]+=f[i]*g[j]
  return r

def shrink(f, c):
    """
    Implements the 'shrinking' of a probability function in the amount c
    """
    return [sum(f[:c+1])]+f[c+1:]

def AF(r,t,c):
  """
  Implements the 'Addition From' operation, which provides the response
  time of Job_a, including the interference of Job_b, being
    r: provisional response time of Job_a
    t: arrival time of Job_a, with respect of arrival time of Job_b
    c: execution time of Job_b
  """
  if t>len(r): return r # No interference is possible
  return r[:t+1]+convolve(r[t+1:], c)

# Utility functions
# ============================================================
def difference(f,g):
    """
    Computes the quadratic difference between two probability functions
    """
    m=min(len(f), len(g))
    M=max(len(f), len(g))
    if (M==len(f)): longest=f
    else: longest=g
    r=0.0
    for i in range(m): r=r+abs(f[i]-g[i])
    for i in range(m,M):
       r=r+longest[i]
    return r

def average(PFs):
    """
    Given a list of probability functions, it returns the average
    """
    maxsize=max(map(len, PFs))
    for pf in PFs:
      pf.extend([0]*(maxsize-len(pf)))
    r=[]
    for i in range(maxsize):
```

```
63      r.append(sum([x[i] for x in PFs]))
    return [x/len(PFs) for x in r]
65
def PDF(pf):
67      """
    Given a probability function (pf), it returns the cumulative
69      distribution function (pdf).
    """
71      pdf=[]; acum=0
    for v in pf:
73          acum+=v
        pdf.append(acum)
75      return pdf

77  def PF(d):
    """
79      Given a PF implemented in a dictionary (pairs of time:probability)
    returns a list (array) of probabilities indexed by time. For
81      example, the dictionary {0:0.2, 7: 0.3, 15:0.5} would produce
    the list [0.2, 0, 0, 0, 0, 0, 0, 0.3, 0, 0, 0, 0, 0, 0, 0, 0.5]
83      with 16 elements.
    The first form is more suitable for input disperse functions, but
85      the second is more suitable for the analysis.
    """
87      m=max(d.keys())
    lista=[]
89      for i in range(0,m+1):
        if d.has_key(i): lista.append(d[i])
91          else: lista.append(0)
    return lista
93
# Input/Output functions
95  # ============================================================
def PF_load(filename):
97      """
    Given the name of a file, this functions reads it an returns a
99      list (array) in which the index is the time and the value
    the probability.
101     The file is expected to contain in each line a pair 'time probability'
    separated by tab ('\t')
103     """

105     d={}
    fil=open(filename, "r")
107     for l in fil:
        c=l.split('\t')
109         if len(c)==2:
          d[int(c[0])]=float(c[1])
111     return PF(d)

113 def PF_save(pf, filename, decimal_places=15):
    """
115     Saves the probability function pf in a file, in a format suitable
    for plotting with gnuplot or tikz.
117     """
    format="%%10i\t%%2.%df" % decimal_places
119     fil=open(filename, "w")
    for i in range(len(pf)):
121         print >> fil, format % (i, pf[i])

123 # END of module
```

A.2 Execution

In order to test the python code, you need only put both files in the same directory and issue the command

```
$ python example-edf.py
```

provided that you have the python interpreter installed.

References

[1] L. Abeni and G. Buttazzo. Stochastic Analysis of a Reservation Based System. In *Proc. of the 9th Int. Workshop on Parallel and Distributed Real-Time Systems*, Apr. 2001.

[2] A. K. Atlas and A. Bestavros. Statistical Rate Monotonic Scheduling. In *Proc. of the 19th IEEE Real-Time Systems Symposium*, pages 123–132, Dec. 1998.

[3] J. L. Díaz, D. F. García, K. Kim, C.-G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella. Stochastic Analysis of Periodic Real-Time Systems in a Real-Time System. In *Proc. of the 23rd IEEE Real-Time Systems Symposium*, pages 289–300, Austin, Texas, Dec. 2002.

[4] J. L. Díaz, J. M. López, M. García, A. M. Campos, K. Kim, and L. Lo Bello. Pessimism in the stochastic analysis of real-time systems: Concept and applications. In *Proc. of the 25rd IEEE Real-Time Systems Symposium*, Lisboa, Portugal, Dec. 2004.

[5] M. K. Gardner. *Probabilistic Analysis and Scheduling of Critical Soft Real-Time Systems*. PhD thesis, University of Illinois, Urbana-Champaign, 1999.

[6] M. K. Gardner and J. W. S. Liu. Analyzing Stochastic Fixed-Priority Real-Time Systems. In *Proc. of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Mar. 1999.

[7] K. Kim, J. L. Díaz, L. Lo Bello, J. M. López, C.-G. Lee, and S. L. Min. An exact stochastic analysis of priority-driven periodic real-time systems and its approximations. *IEEE Trans. on Computers*, 54(11):1460–1466, 2005.

[8] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.-S. Liu. Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times. In *Proc. of the Real-Time Technology and Applications Symposium*, pages 164–173, Chicago, Illinois, May 1995.