DMAC: Deadline-Miss-Aware Control

Paolo Pazzaglia
Scuola Superiore Sant'Anna, Pisa, Italy
Department of Automatic Control, Lund University, Sweden
paolo.pazzaglia@sssup.it

Claudio Mandrioli
Department of Automatic Control, Lund University, Sweden
claudio.mandrioli@control.lth.se

Martina Maggio
Department of Automatic Control, Lund University, Sweden
martina.maggio@control.lth.se

Anton Cervin
Department of Automatic Control, Lund University, Sweden
anton.cervin@control.lth.se

Abstract

The real-time implementation of periodic controllers requires solving a co-design problem, in which the choice of the controller sampling period is a crucial element. Classic design techniques limit the period exploration to *safe* values, that guarantee the correct execution of the controller alongside the remaining real-time load, i.e., ensuring that the controller worst-case response time does not exceed its deadline. This paper presents DMAC: the first formally-grounded controller design strategy that explores shorter periods, thus explicitly taking into account the possibility of missing deadlines. The design leverages information about the probability that specific sub-sequences of deadline misses are experienced. The result is a *fixed* controller that on average works as the ideal clairvoyant time-varying controller that knows future deadline hits and misses. We obtain a safe estimate of the hit and miss events using the *scenario theory*, that allows us to provide probabilistic guarantees. The paper analyzes controllers implemented using the Logical Execution Time paradigm and three different strategies to handle deadline miss events: killing the job, letting the job continue but skipping the next activation, and letting the job continue using a limited queue of jobs. Experimental results show that our design proposal – i.e., exploring the space where deadlines can be missed and handled with different strategies – greatly outperforms classical control design techniques.

2012 ACM Subject Classification Computing methodologies → Computational control theory; Computer systems organization → Embedded software; Software and its engineering → Real-time systems software; Theory of computation → Stochastic control and optimization

Keywords and phrases Weakly-Hard Real-Time Systems, Deadline Miss Handling, Control Design.

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2019.17

1 Introduction

Controllers are often executed alongside other tasks in a real-time platform, demanding that the scheduler ensures the timely execution of both the controller and the real-time workload that the platform should execute. Controllers can be designed taking into account resource limitations and scheduling constraints [16, 54, 55].

Studying the optimal design of a control task to be run alongside a given real-time workload can be considered an instance of the general problem of *composability*. Composability is the capability to integrate new functionalities into a preexisting system. This issue is particularly relevant in the automotive field, where the production of new vehicles requires a tight coupling of new software together with legacy code, with minimal adjustment of the original structure. In general, adding a new control task to a given taskset implies combining requirements

31st Euromicro Conference on Real-Time Systems (ECRTS 2019).
Editor: Sophie Quinton; Article No. 17; pp. 17:1–17:24
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that come from both control theory and real-time implementation. These requirements are different and often conflicting. As an example, selecting a high execution rate for the controller improves the control performance, but at the same time limits the guarantees on the timely completion of the control task code and forces the engineers to take into account overruns [14, 41]. Moreover, minimizing the monetary cost of the final system is an ever-present priority and over-provisioning resources is usually not a viable solution.

Timing constraints in real-time systems are modeled as *deadlines*, i.e., a threshold that the execution time of each task instance (*job*) should respect. We refer to a job that successfully completes its execution before the corresponding deadline as a *deadline hit* event. If the job could not terminate its execution before that deadline instant, we say that it *missed its deadline*. In *hard* real-time systems, missing a deadline has been always seen as a risk that must be avoided, with possibly catastrophic consequences. In reality, a limited number of deadline misses is an acceptable condition for many cyber-physical and control systems, since well-designed controllers often expose intrinsic *robustness* to timing non-idealities. Recently, researchers have then tried to formally relax deadline constraints, introducing the *weakly hard* real-time system paradigm [7] to describe the case where tasks are allowed to miss a limited number of deadlines. However, often control engineers lack information about the timing behavior of the control task and the taskset structure. Understanding how a control loop behaves under deadline misses may open the door to new and better control designs.

Inspired by this challenge, in this paper we tackle the problem of designing a controller for a generic physical plant, while exploring a range of periods which have historically been avoided for co-design: we are here interested in those period values that are *shorter* than the worst-case response time of the task, thus neglecting the common hypothesis of hard deadlines. Our design problem is to run the controller alongside a preexisting taskset. Tasks are described with probabilistic execution times, ranging from a best case to a (rare) worst case value. By leveraging the flexibility of robust control design techniques, we here propose a novel method for creating an optimal fixed controller, the *Deadline-Miss-Aware Control* (DMAC), which can be implemented in a real-time task that may miss some deadlines.

The DMAC design takes into account how the controlled system behaves when different patterns of hit and missed deadlines occur. For robustness, DMAC considers a safe (pessimistic) probability of deadline miss events. Lack of scalability impedes the computation of deadline miss probabilities analytically. However, bounds are extremely pessimistic and would not aid the control design method. To overcome this limitation, we obtain an estimate of deadline miss occurrence simulating the schedule execution, drawing execution times (for all the tasks) from the corresponding probability distributions. A robust control tool, the *scenario theory* [11], provides the means to select the worst-case sequence of misses and hits from the simulations. Leveraging the scenario theory, our approach allows us to provide probabilistic guarantees for worst-case conditions both in terms of the probability of not having taken into account conditions that will eventually manifest, and in terms of the design confidence. We obtain a controller which is optimal and robust to worst-case conditions.

The analysis presented in this paper considers three different strategies for handling deadline misses: kill the job that missed the deadline, let it continue and skip the next job, or let all jobs continue until completion, but limiting the ready queue to the most recently activated among the pending jobs (in addition to completing the one that missed the deadline). We implement the controller following the Logical Execution Time (LET) paradigm [28]. To the best of our knowledge, this is the first attempt to design an optimal controller for a real-time system that is aware of deadline misses and miss-handling strategies.

2 Methodology

The aim of DMAC is to provide the first control synthesis method that is *robust* both with respect to deadline misses and with respect to the strategy used to handle them. Our control design leverages knowledge of the probability that different sequences of deadline hits and misses may occur, and produces a fixed controller that is (on average) optimal with respect to a defined cost function. We obtain such knowledge by formulating a chance constrained optimization problem in a probabilistic framework, and obtaining guarantees on both the probability of neglecting important information and the confidence in the design.

2.1 Overview and Terminology

We present here an overview of the approach adopted for the control design and evaluation. For the application of the approach we rely on the following input data:

- ε: The user selects a defined value of ε, that represents a worst case bound on the probability of carrying out the control design operations neglecting important information. We rely on the generation of sequences of deadline hits and misses, from which we select the worst case and design our controller to be robust with respect to such worst case. While ε can be selected to be as small as possible, we still need to accept a certain small probability that the next generated sequence would be worse than the worst case generated up to the current one.
- $1-\beta$: The user selects a value for the confidence that one can have in the probabilistic guarantee that the approach is providing. The value of $1-\beta$ is determined together with the value of ε, to indicate that the approach is based on a confidence $1-\beta$ that the probability ε is the true probability of missing important information.
- Γ: The taskset that our design is targeting. We assume that additional (hard real-time) load is run alongside the controller task.
- n_{job}: Our control design is based on extracting timing behavior from simulations of a certain number of control jobs. The length of the sequences used for the controller design can be chosen depending on physical parameters, for example assuming that after a certain number of jobs the controller has settled. We recommend to select a value that contains at least a few hyperperiods, to capture chain effects if they happen.
- J_{seq}: The cost function that is used to evaluate the produced sequences to select the worst-case sequence for the controller design.
- ξ: The strategy used to handle a deadline miss. We consider three different strategies: killing the job that missed the deadline, letting it continue and skipping the next job, or letting it continue and enqueuing the next job (up to a maximum of one enqueued job at any point in time).
- J_{ctl}: The cost function that is used to evaluate the controller behavior and compare the different deadline miss handling strategies.

Figure 1 visually shows the different steps, inputs and outputs. As shown in the figure, our approach feeds the probability bounds (ε and $1-\beta$) to the "Scenario Theory" [11] block. The scenario theory is used in control for the design of robust controllers to handle uncertainty that is *a priori* unpredictable in the disturbance values and in the system model. In this paper we reinterpret the scenario results to enable a control synthesis strategy that uses deadline hits and misses information and provides (probabilistic) guarantees.

The scenario theory is a formal tool that determines how to analyze experimental data. In particular, we schedule our taskset extracting execution times from the corresponding

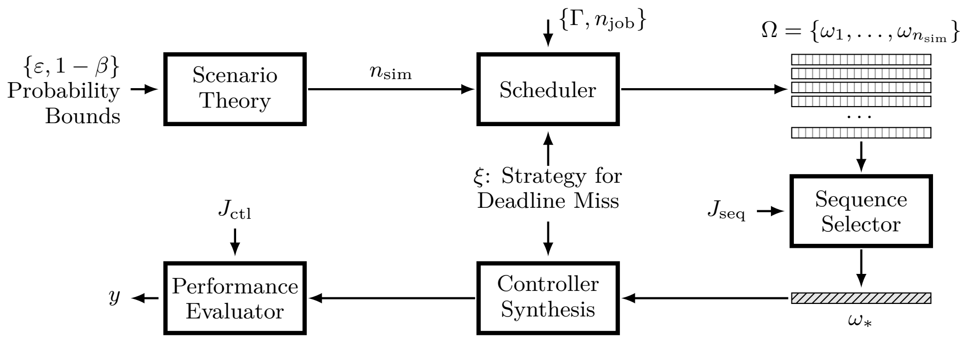

Figure 1 Approach Overview.

probability distributions that are known in the Γ taskset. The theory provides us with information on how many experiments (scheduling simulations) we should execute in order for the probability that unforeseen circumstances are worse than the gathered data to be lower or equal to ε with confidence $1-\beta$. We denote the number produced by the scenario theory with n_{sim}. For each of the n_{sim} experiments, we randomly sample the probability distributions of the task execution times, to generate a set $\Omega = \{\omega_1, \ldots, \omega_{n_{\text{sim}}}\}$ scheduling sequences, in which the control task executes for n_{job} times, using strategy ξ to handle the deadline misses. Using our scheduler, we record sequences of deadline hits and misses.

We evaluate each of these sequences with a cost function J_{seq}, identifying the worst sequence ω_*, from the control perspective. From this sequence we extract the probability of deadline hits and misses for each of the n_{job} instances of the control task and the joint probability distribution for each sequence of hits and misses needed for the control design. The controller synthesis block uses the extracted information for the control strategy design. The generated controller is then evaluated when the taskset is executed and the controller is connected to the real plant, using a cost function J_{ctl}, which allows us to compare the performance of different deadline management strategies. We can then determine the best deadline management strategy and control period for the system under analysis.

As output of our approach we obtain y, the evaluation of each tested strategy ξ for the specific problem. As a by-product, we also obtain the set of sequences Ω. If we are not satisfied with our controller behavior, we can analyze the set of sequences to understand how to improve the control performance (i.e., for example optimize a certain task in the taskset).

Paper Organization

In the following, Section 3 discusses the model used for both the plant and the taskset, and Section 4 describes the behavior of the system using different deadline miss handling strategies. Section 5 presents the control design approach. In Section 6 we present the framework that we use to obtain probabilistic information about the scheduler behavior, and the scenario theory. In Section 7 we show our experimental setup and the evaluation criteria, and present our results. Section 8 discusses related work, and Section 9 concludes the paper.

3 System Model and Problem Definition

This section introduces the models used in the paper. Section 3.1 describes the model of the taskset executing on the hardware. Section 3.2 discusses the models of plant and control task. Finally, Section 3.3 introduces the three strategies used to handle deadline misses.

3.1 Taskset Model

In this paper, a real-time workload Γ is defined as the union of a (given) set of generic hard real-time periodic tasks, plus a real-time control task τ_d, which is the target of our design, i.e., $\Gamma = \Gamma' \bigcup \{\tau_d\}$. In this description, Γ' is a set of N_T periodic tasks, i.e., $\Gamma' = \{\tau_1, \tau_2 \ldots, \tau_{N_T}\}$ and τ_d is an additional periodic task that contains our controller operation. We assume that each task τ_i is independent from the others and released synchronously at a given starting instant. The tasks are scheduled using a fixed priority scheduling policy (e.g. Rate Monotonic) with preemption, and the indexing reflects their priority ordering, i.e. τ_i has higher priority than τ_j if $i < j$. In our design problem, τ_d is the task with the lowest priority.

Each task is characterized by a tuple of parameters, $\tau_i = (\mathcal{C}_i, f_i^{\mathcal{C}}, D_i, T_i)$. Here, $\mathcal{C}_i$ is a random variable that represents the task execution time, while $f_i^{\mathcal{C}}(c)$ is its probability density function, i.e. $\forall c \in \mathbb{N}$, $f_i^{\mathcal{C}}(c) = \mathbb{P}\{\mathcal{C}_i = c\}$; D_i and T_i are deterministic values, representing respectively the task deadline and period. In accordance with the literature on real-time applications for control systems, task periods are chosen among a limited set of possible values, typically related to physical requirements of the control task [32, 39].

For each task τ_i, we consider a discrete probability distribution $\mathcal{C}_i$ with N_i integer values, ranging between a Best Case Execution Time (BCET) $C_i^{\min}$ and a Worst Case Execution Time (WCET) $C_i^{\max}$. Furthermore, we consider tasks that behave well in most cases, i.e., tasks whose probability density functions are skewed towards lower values. In fact, while our approach can be applied to systems with generic probability density functions, we want to capture tasks which experience occasional faulty conditions. This choice is in agreement with most works that analyze execution time distributions for real-time tasks [53]. We will generally refer to the utilization of taskset Γ' as the *worst-case* utilization, i.e. $U_{\Gamma'} = \sum_{i=1}^{N_T}(C_i^{\max}/T_i)$.

We denote each periodic instance of $\tau_i \in \Gamma$ with the term job, and define it as $J_{i,k}$, with $k = 1, 2, \ldots$ representing the job index and i representing the task index. For every job $J_{i,k}$, $a_{i,k}$ denotes the activation instant, and $a_{i,k+1} - a_{i,k} = T_i$. Since we are considering synchronous release conditions, $\forall i$, $a_{i,0} = 0$ holds. In the following, $\mathcal{R}_{i,k}$ represents the random discrete variable that models the response time of $J_{i,k}$. The Worst Case Response Time (WCRT) of task τ_i is denoted as R_i^W and computed with standard techniques [33], by considering the condition where every task experiences its WCET. Similarly, the Best Case Response Time (BCRT) [43] is introduced as R_i^B and computed considering that every job executes with its BCET. Finally, in this work all tasks τ_i in Γ' are *schedulable*, i.e. $R_i^W \leq D_i$ for each τ_i. However, this hypothesis will not be required for τ_d. We will only assume that at least one job of τ_d respects its deadline, i.e. $R_d^B \leq D_d$.

3.2 Plant and Controller Model

The plant to be controlled by τ_d is described as a linear time invariant, multi-input multi-output system in continuous time. In line with standard assumptions, we assume the plant to be controllable and the state to be fully measurable. The plant dynamics is described as

$$\dot{\mathbf{x}}(t) = A_c\,\mathbf{x}(t) + B_c\,\mathbf{u_c}(t) + \mathbf{v_c}(t). \tag{1}$$

In Equation (1), every element in bold represents a vector, while A_c and B_c are the constant matrices that encode the dynamic evolution of the system. The term $\mathbf{x}(t)$ denotes the system state vector and $\dot{\mathbf{x}}(t)$ its time derivative. The term $\mathbf{u_c}(t)$ is the vector that contains the control signals. The vector $\mathbf{v_c}(t)$ represents the plant disturbance, modeled as white noise with known covariance matrix R_c. The goal of the control is to minimize a cost function,

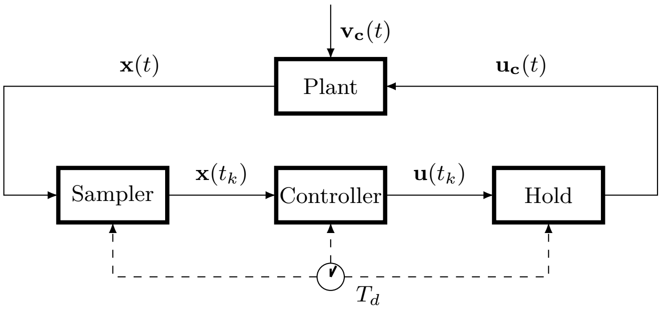

Figure 2 Plant and controller with time-triggered sampler and hold devices.

defined as the mean value of a quadratic function of the state vector and the control vector:

$$J_{\text{ctl}} = \mathbb{E}\left\{\int \mathbf{x}^{\mathrm{T}}(t)Q_{1c}\mathbf{x}(t) + \mathbf{u_c}^{\mathrm{T}}(t)Q_{2c}\mathbf{u_c}(t)\right\}. \tag{2}$$

Here, $\mathbb{E}$ indicates the expected value, while Q_{1c} and Q_{2c} are constant positive semidefinite matrices and design parameters of the controller. They represent the trade-off between regulating $\mathbf{x}(t)$ to zero and the cost of using the control signal $\mathbf{u_c}(t)$. This cost function is used both as a controller design objective and for performance evaluation of the control task.

The plant is connected to the controller via time-triggered sampler and hold devices as shown in Figure 2. The behavior of these devices can be modeled as a dedicated task that reads and writes data with zero execution time and highest priority. The plant state is sampled every T_d time units, implying $\mathbf{x}(t_k) = \mathbf{x}(kT_d)$. The control job $J_{d,k}$ is released at the same instant, i.e. $a_{d,k} = kT_d$, and the sensor data $\mathbf{x}(t_k)$ is immediately available to it. Based on the state measurement, the controller computes the feedback control action $\mathbf{u}(t_k)$.

As an hypothesis, our control task τ_d executes under the Logical Execution Time paradigm. Indeed, the job $J_{d,k}$ computes the control output using $\mathbf{x}(t_k)$ but makes it available to the actuator only at the first deadline instant after the termination of its execution. The control actuation is then held constant until the *next* update. This means that, if all jobs finish before their deadline, the following equation holds:

$$\mathbf{u_c}(t) = \mathbf{u}(t_k), \qquad a_{d,k} + D_d \leq t < a_{d,k+1} + D_d. \tag{3}$$

The execution time of the control task τ_d is given as a random variable with known probability density function, and is treated equivalently to any other task in Γ'. On the contrary, the deadline D_d and period T_d of the control task τ_d are part of the design. Being a LET task, we restrict our analysis to the implicit deadline case ($D_d = T_d$), although in principle the approach in the paper can be applied to other relative deadlines (and corresponding output times). We further assume that the execution time properties of the controller do not change with different periods and different controller parameters (since only the values of some parameter are modified but the operations done by the control task are the same).

In the paper, τ_d is not treated as a hard-deadline task. On the contrary, we actively look for those values of T_d such that the resulting task may miss some deadline with probability greater than zero, but still being able to guarantee a good control performance. This is to increase the system utilization, and consolidate workload on one single core. In Section 5, we present how to properly characterize the timing behavior of the controller and its synthesis.

▶ Remark 3.1. In this paper, we work under the assumption that τ_d is the task with the lowest priority. If other tasks with priority lower than τ_d do exist, the design proposed hereafter is still valid in principle, since those tasks cannot interfere with τ_d. However, if this is the case, the range of possible values of T_d should be tied with the schedulability guarantees for the lower priority tasks. We reserve to analyze this more general case as a future work.

3.3 Handling Deadline Misses

In classic control design, the control task behavior is assumed to be strictly periodic, or at least periodic with limited reponse time jitter [16]. To provide an implementation enforcing periodicity for a controller on a real-time platform, one usually selects a period for the control task that is greater or equal than the task's WCRT. This approach is safe but can be very pessimistic. In fact, WCRT conditions may be extremely rare. Selecting the period with the mentioned constraint could limit the achievable control performance, since longer sampling periods in general mean worse disturbance rejection and smaller stability margins [15].

The approach presented in this paper explores the possibility of designing a control task with periods smaller than its WCRT, i.e., $T_d < R_d^W$, thus greatly extending the design space. We remark that, with $T_d \geq R_d^W$ there are no deadline misses, and standard approaches for the control design can be used [4, 29, 55].

Choosing $T_d < R_d^W$ implies the risk that the control task will miss some deadlines. A deadline miss is a timing violation that can produce unbounded response times, due to self-pushing [47], and therefore it should be properly handled. In this work, we consider three different strategies to handle deadline misses, that have previously been explored in the control community [14]: **(i)** to *kill* the job that has not completed at the deadline, **(ii)** to let the current job continue but *skip the next* job(s), and **(iii)** to let the job continue, placing the next job(s) in a *queue of length one*. In more detail, these strategies behave as follows:

- Kill: A control job that is not able to terminate within its deadline is dropped at the deadline instant. When a job is killed, its (partial) computation is discarded and no output is produced. We assume that this dropping mechanism has negligible overhead and internal states of the controller are not altered by the partial computation.
- Skip-Next: A control job that is not able to terminate within its deadline is allowed to continue until completion. However, whenever the active job exceeds a deadline, the next instance of the control job is not activated (skipped). This is based on the idea that completing a job that has already started is preferred to starting a new one and incurring the risk that the computation runs longer than the deadline again.
- Queue(1): A control job that is not able to complete its execution within its deadline is allowed to continue its execution, while the following jobs are put in a queue that can contain a single element. Thus, at the activation of a new job, if there is already an active instance, the new job is enqueued, overwriting the currently existing job in the queue. Only the most recently arrived job is stored in the queue and is activated as soon as the current job completes.

An example of a schedule under the three strategies is presented in Figure 3, where the odd jobs are shown in dark gray, while the even jobs are shown in light gray. With the Kill strategy, the first job is killed at its deadline, before completing its execution. With the other two strategies, the job is given additional time in the second period, and is therefore able to complete, albeit running over time. With the Skip-Next strategy, the second job is not started, since there is an active control job that has not terminated its computation. With the Queue(1) strategy, the second job also runs over time, due to interference. In general, the Kill and Skip-Next strategies avoid self-interference conditions. This is not true

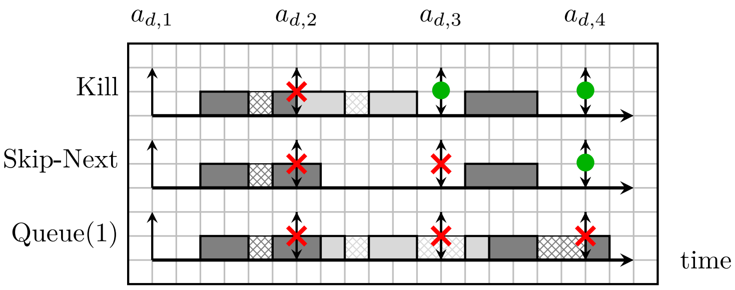

Figure 3 Schedule example using the three proposed strategies to handle deadline misses. Those jobs that missed a deadline (or are skipped) are marked with a red cross on their deadline, while a green dot identifies deadline hit events.

for the Queue(1) strategy. However, Queue(1) may be seen as a particular case of finite buffer strategy [1], where the freshest job of the queue is always preferred, discarding the old one that has not yet started. This choice helps reducing the amount of self-interference and avoids unbounded response times. In practice, a job that is delayed more than one period by self-interference is skipped and the next one is put in the ready queue.

A sequence of consecutive control jobs may contain a certain number of jobs that are not actively contributing to the actuation $\mathbf{u}(t)$. This happens either with jobs that are terminated before completing their execution (killed) or with jobs not executed at all (skipped). For those jobs, a proper response time value may not be defined. Moreover, under Queue(1) strategy, it may happen that the output of a job which completes its execution after missing one or more deadlines is *overwritten* by the next job, if the latter completes before the same deadline. We therefore define the set of jobs that produce an output control that is actually provided to the physical plant, as the set of *valid* control jobs.

▶ **Definition 3.2** (Valid control job). *A* valid *control job ν is a job that successfully completes its execution and whose generated output is not overwritten before the next deadline instant.*

For each time interval $[0, t)$, we show that is possible to extract the ordered sequence of v valid jobs, defined as $S = \{\nu_1, \nu_2, ..., \nu_v\}$ (where the index does not count the passing of time) and the relation $v \leq \lceil t/T_d \rceil$ trivially holds. The sequence of valid jobs depends on the strategy used to handle deadline misses, and will be described in Section 4. Our control design should be robust not only to the possibility of missing deadlines, but also to the different pattern of delays that are produced depending on the strategy used to handle the miss event. In the following section, we discuss how this affects the control task behavior.

4 Controller Behavior with Deadline Misses

In theory, choosing a shorter period allows the discrete-time controller to achieve better control performance [5]. However, real-time constraints become harder to satisfy, due to the increased interference from higher priority tasks. Since we are targeting periods shorter than the WCRT, the probability of missing a deadline for the control task is greater than zero. A controller designed with standard techniques and subject to overrun may still display some intrinsic robustness when experiencing occasional deadline misses [14, 41]. However, the controller performance depends strongly on the deadline miss handling strategy, and may become unacceptable when the probability of missing a deadline is too high. Here we analyze the timing properties of a control task subject to deadline misses. We show that

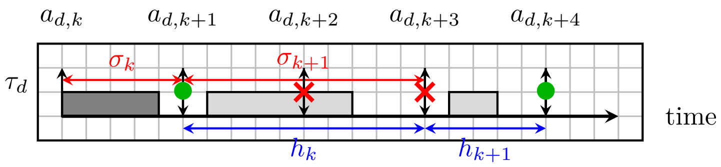

Figure 4 Example of delay and hold values for Skip-Next strategy.

only two parameters are needed to fully characterize the miss impact. We build the rules for computing them and extract the sequence of valid control jobs.

4.1 Defining Delay and Hold

Missed deadlines invalidate one of the basic hypothesis of control theory, which is the periodicity of the output pattern [41]. In this work, we exploit the knowledge of deadline misses directly in the control design step. For this purpose, we need to characterize how deadline misses affect the control performance. We fully describe the effect of deadline misses of LET-based controllers with two parameters, named respectively *delay* and *hold* interval.

▶ **Definition 4.1** (Delay σ_k)**.** *The* delay σ_k *experienced by a control job* $J_{d,k}$ *is defined as the time interval between the activation instant of the job* $a_{d,k}$ *and the instant where its control output is made available to the actuator.*

In other words, the delay σ_k represents the time from sampling the plant state until updating the control signal. Using the above formulation, σ_k can only be properly defined for jobs that correctly complete their execution: if a job is killed or skipped, no delay information can be extracted, since its computation does not properly finish. We will refer to this condition as an *undefined* delay, represented with the symbol ∞. From a control perspective, the delay experienced by each (completed) job must be compensated accordingly by a predictor that computes the expected state at the output instant $(t_k + \sigma_k)$, using the knowledge of the current state $\mathbf{x}(t_k)$ and the control output(s) active in that time span.

▶ **Definition 4.2** (Hold interval h_k)**.** *Given a control output computed by* $J_{d,k}$ *and available at the actuator for the first time at* $t_k + \sigma_k$*, the* hold interval h_k *is the time interval between* $t_k + \sigma_k$ *and the first instant where a new control output is made available.*

In other words, the hold interval h_k indicates the lifetime of the control signal computed by the k-th controller job and thus represents the time interval in which the computed control signal is held constant. Similarly to the delay, the definition of h_k is meaningful only for jobs that correctly complete their execution. If job $J_{d,k}$ is killed or skipped, the hold interval is *undefined* and will be represented with the symbol ∞. Moreover, if job $J_{d,k}$ correctly completes its execution, but its output is overwritten by the output of job $J_{d,k+1}$ before being used, we will assign a hold value $h_k = 0$. Figure 4 shows an example of delay and hold intervals for a sequence of four jobs using the Skip-Next strategy.

4.2 Computing Delay and Hold

Under ideal timing conditions—i.e., when all the deadlines of the control task are hit—the control signal produced in one period is always applied in the next period. The controller should thus compensate for a fixed delay $\sigma_k = T_d$. In this situation, the delay and hold have the same value and they are often not even defined as two different parameters. However, when considering deadline misses, σ_k and h_k may assume different values. Figure 4 shows one such example, where $\sigma_k = 1\,T_d$, $h_k = 2\,T_d$, $\sigma_{k+1} = 2\,T_d$, and $h_{k+1} = 1\,T_d$.

The potential differences between the delay and hold values have several consequences for the control design. First, a predictor designed for a one period delay may produce a value that is incorrect for longer delays. Second, a control signal calculated for a short hold interval could be too aggressive if applied in longer intervals. Lastly, the resulting delay–hold pattern may change across multiple control activations and depends heavily on the deadline miss handling strategy that has been chosen.

Computing the values of σ_k and h_k for each $J_{d,k}$ in a schedule is then crucial for the control design process. In fact, knowing in advance the values of delay and hold interval of each job, enables the design of an optimal *time-varying* controller, that for each control job selects how to compensate the particular combination of σ and h for the current and following jobs. This however would require a *clairvoyant* controller, that is not practically realizable. Here we extract the possible pairs $(\sigma_k,\, h_k)$ that may happen in a given scheduling sequence, and their associated probability, to design a fixed controller that behaves as close as possible to the ideal unrealizable one. We discuss the controller synthesis in Section 5.

Knowing σ_k and h_k for a given job $J_{d,k}$, it is also possible to determine whether the k-th control job is *valid.* This corollary follows from the definition of delay and hold interval:

▶ **Corollary 4.3.** *A job $J_{d,k}$ is* valid *if and only if it is possible to define both its delay σ_k and hold interval h_k (i.e., they are finite numbers) and if the hold interval is greater than zero.*

As a consequence, the pairs $(\sigma_k,\, h_k)$ can be leveraged to extract the set of ordered *valid* jobs, which are the ones effectively used for building the controller. We now discuss how to compute σ_k and h_k for each $J_{d,k}$ with the different miss-handling strategies. First of all, it is worth noting that the definition of σ_k is strictly related to the notion of response time of job $J_{d,k}$. In fact, the control output computed by $J_{d,k}$ is dispatched to the actuator at the first control activation (i.e. the closest incoming deadline) that follows the termination of $J_{d,k}$. The delay σ_k of $J_{d,k}$ can then be computed (for each strategy) as follows:

$$\sigma_k = \begin{cases} \lceil \mathcal{R}_{d,k}/T_d \rceil \, T_d & \text{if } J_{d,k} \text{ completes} \\ \infty & \text{otherwise.} \end{cases} \tag{4}$$

Trivially, the maximum value for σ_k is $\bar{\sigma} = \lceil \mathcal{R}_d^W / T_d \rceil T_d$. While extracting σ_k requires only the knowledge of $J_{d,k}$, in order to compute the value of the hold interval h_k it is necessary to know the behavior of the control jobs executing after $J_{d,k}$, until the release of a new control update. In practice, this means that only a finite number of sub-sequences needs to be checked for characterizing all possible combinations of $(\sigma_k,\, h_k)$. Below, the equations for computing h_k for each strategy are presented in detail.

4.2.1 Hold Interval with Kill Strategy

Using the *Kill* strategy, the control job either finishes within one period or it is killed at its deadline. An arbitrary sequence of deadline misses may happen between two jobs that complete successfully. Denoting with $\lambda_{k,\text{Kill}}$ the number of consecutive jobs that miss their deadline after $J_{d,k}$, the hold interval associated to $J_{d,k}$ is computed according to

$$h_k = \begin{cases} (\lambda_{k,\text{Kill}} + 1) \cdot T_d & \text{if } J_{d,k} \text{ completes} \\ \infty & \text{otherwise (if } J_{d,k} \text{ is killed).} \end{cases}$$

If $J_{d,k}$ has been killed, h_k is not defined. Note that if some weakly hard constraint is known for τ_d, the values of $\lambda_{k,\text{Kill}}$ to check may be upperbounded with the maximum possible number of consecutive deadline misses of that task.

4.2.2 Hold Interval with Skip-Next Strategy

Using the *Skip-Next* strategy, no new job may be activated while the active one is executing. Denoting with $\lambda_{k,\text{Skip-Next}}$ the number of skipped jobs that directly follows $J_{d,k}$, the hold interval of a job $J_{d,k}$ can then be computed as

$$h_k = \begin{cases} \sigma_{k+1+\lambda_{k,\text{Skip-Next}}} & \text{if } J_{d,k} \text{ completes} \\ \infty & \text{otherwise } (J_{d,k} \text{ is skipped}). \end{cases}$$

If $J_{d,k}$ is skipped, h_k is not defined. Intuitively, this means that for the Skip-Next strategy, the hold value of one completed job is equal to the delay of the subsequent job that completes (i.e., of the next valid job). In the example of Figure 4, job $J_{d,k}$ terminates correctly, while $J_{d,k+1}$ does not complete before its deadline. $J_{d,k+2}$ is then skipped. The hold value h_k is therefore equal to the delay σ_{k+1} which is $2\,T_d$ since the job has an overrun. The hold value h_{k+1} is equal to the delay of the next completed job $J_{d,k+3}$, i.e., $h_{k+1} = \sigma_{k+3} = T_d$. The values that $\lambda_{k,\text{Skip-Next}}$ may assume are upperbounded by $\lceil R_d^W / T_d \rceil - 1$.

4.2.3 Hold Interval with Queue(1) Strategy

Using the *Queue*(1) strategy, if a job misses its deadline, two scenarios may happen: it completes before the deadline of $J_{d,k+1}$, or it finishes later, then some of the subsequent jobs is skipped. In both those cases, however, the instant where the control output is published to the actuator falls exactly one period (T_d) after the activation of the next valid job. Denoting with $\lambda_{k,\text{Queue}(1)}$ the number of (eventually) skipped jobs that directly follows $J_{d,k}$, the hold value h_k for job $J_{d,k}$ is computed as follows:

$$h_k = \begin{cases} \sigma_{k+1} & \text{if } J_{d,k} \text{ hits its deadline} \\ \sigma_{k+1+\lambda_{k,\text{Queue}(1)}} - T_d & \text{if } J_{d,k} \text{ misses its deadline} \\ \infty & \text{otherwise (skipped).} \end{cases}$$

If $J_{d,k}$ is skipped because it is removed from the queue due to a subsequent activation, h_k is not defined. Note that if $J_{d,k}$ misses and $J_{d,k+1}$ hits its deadline—i.e., if both the k-th and the $k+1$-th control jobs complete before during the $k+1$-th period—then $\sigma_{k+1} - T_d = 0$, and the control signal produced by $J_{d,k}$ is never actuated. Finally, values of $\lambda_{k,\text{Queue}(1)}$ are upperbounded by $\lceil (R_d^W - T_d)/T_d \rceil - 1$.

5 Synthesis of Deadline-Miss-Aware Controllers

Standard digital control design assumes that samples are taken regularly and that there is a (most likely known and constant) delay from sampling to actuation [5]. When deadlines are missed, the actual hold and delay intervals will deviate from the assumed values, as explained in the previous section. This *control jitter* leads to degraded performance, and, in extreme cases, even to instability of the control loop [16]. With some knowledge about the jitter, however, it is possible to synthesize a controller that partially compensates for the timing irregularities. We outline two variants of our Deadline-Miss-Aware Control designs below.

5.1 Clairvoyant Controller Synthesis

The controlled system evolution can be derived by sampling the plant only at the update instants of each valid job ν_n, i.e. at the time where the control output produced by ν_n is provided to the actuator. With a slight abuse of notation we will refer hereafter to the

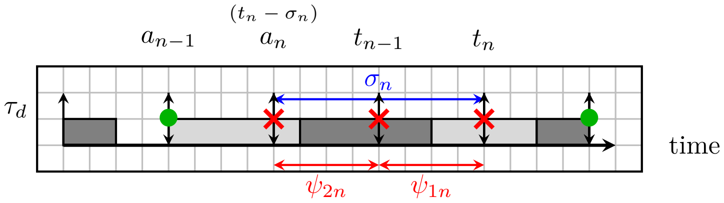

Figure 5 Example of ψ_{1n} and ψ_{2n}.

pair of delay and hold relative to ν_n as (σ_n, h_n), while its activation instant is a_n. The update instant of the control output produced by ν_n can then be defined as $t_n = a_n + \sigma_n$. Moreover, the relation $t_{n+1} = t_n + h_n$ trivially holds. For each valid control job ν_n in sequence $S = \{\nu_1, \nu_2, ..., \nu_v\}$, the state evolution can be calculated as

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n + h_n) = A(h_n)\mathbf{x}(t_n) + B(h_n)\mathbf{u}(t_n) + \mathbf{v}(t_n), \tag{5}$$

where $\mathbf{x}(t_n)$ is the state measurement sampled at time t_n, $\mathbf{u}(t_n)$ the control output released at time t_n, and $\mathbf{v}(t_n)$ a discrete-time model of the plant disturbance. The discrete matrices A and B are sampled from A_c and B_c of (1), respectively, with the step h_n. It is worth noting that different matrices $A(h_n)$ and $B(h_n)$ are created, depending on the possible values of h_n. In fact, a system described in this way behaves as a *switched-linear* system [48]. Computing the matrices can be done with standard procedures for sampled-data systems [5].

If the timing behavior of all jobs was completely known in advance, we would be able to design, by looking offline at the schedule, an optimal time-varying controller that minimizes the cost function (2). We call this a *clairvoyant* controller. The optimal control signal to be applied in the hold interval h_n is given by

$$\mathbf{u}(t_n) = -L_n\mathbf{x}(t_n), \tag{6}$$

where the sequence of feedback gain matrices $\{L_n\}$ are obtained as the solution to a time-varying Riccati equation involving the sequences $\{A(h_n)\}$, $\{B(h_n)\}$, and the sampled equivalents of the cost matrices Q_{1c} and Q_{2c}. The feedback matrices can be calculated off-line and stored in a table for on-line use.

The control law (6) cannot be implemented as it stands, though. The control action must be computed based on a state measurement that is σ_n time units old. Hence the controller must also predict the state from time $t_n - \sigma_n$ to t_n. Note however that in the time interval between $t_n - \sigma_n$ and t_n, the control actuation may not be constant, thus a slightly different modeling is needed. We will refer to the estimate of the state as $\hat{\mathbf{x}}$, which is computed as

$$\hat{\mathbf{x}}(t_n) = A(\sigma_n)\mathbf{x}(t_n - \sigma_n) + A(\psi_{1n})B(\psi_{2n})\mathbf{u}(t_{n-2}) + B(\psi_{1n})\mathbf{u}(t_{n-1}). \tag{7}$$

Here, ψ_{1n} represents the time interval in $[t_n - \sigma_n, t_n]$ when the control actuation of the previous valid job $\mathbf{u}(t_{n-1})$ is held constant, while ψ_{2n} is the (possible) interval where $\mathbf{u}(t_{n-2})$ is active. For the sake of clarity, an example is shown in Figure 5. An operative procedure for computing ψ_{1n} and ψ_{2n} is given as follows:

$$\psi_{1n} = a_n + \sigma_n - (a_{n-1} + \sigma_{n-1}), \qquad \psi_{2n} = a_{n-1} + \sigma_{n-1} - a_n. \tag{8}$$

5.2 Robust Controller Synthesis

The clairvoyant controller has two drawbacks. First of all, it relies on exact knowledge of the execution of the system, ahead of time. This is only possible in very special circumstances.

The other drawback is that it is time varying, which is more complicated to implement and requires extra memory to store the time-varying feedback gain and prediction matrices. A more realistic approach is instead to design a fixed, *robust* controller, based on the statistical properties of the system.

Again starting from the sampled system description (5), we can instead solve a *stochastic* Riccati equation [38] based on the possible values of $A(h_n)$ and $B(h_n)$ and their relative frequency in the schedule during the execution of the system. The control law is then

$$\mathbf{u}(t_n) = -\bar{L}\,\mathbf{x}(t_n), \tag{9}$$

where $\bar{L}$ is a *fixed* gain matrix obtained from the solution to the stochastic Riccati equation

$$\begin{aligned}
\bar{X} &= \mathbb{E}\left\{ \begin{bmatrix} A(h_n)^{\mathrm{T}} \\ B(h_n)^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \bar{S} \begin{bmatrix} A(h_n)^{\mathrm{T}} \\ B(h_n)^{\mathrm{T}} \end{bmatrix} + \begin{bmatrix} Q_1(h_n) & Q_{12}(h_n) \\ Q_{12}(h_n)^{\mathrm{T}} & Q_2(h_n) \end{bmatrix} \right\} \\
\bar{S} &= \bar{X}_{11} - \bar{L}^{\mathrm{T}} \bar{X}_{22} \bar{L} \\
\bar{L} &= \bar{X}_{22}^{-1} \bar{X}_{12}^{\mathrm{T}}.
\end{aligned}$$

This would be the optimal fixed-gain control law if the matrices $A(h_n)$ and $B(h_n)$ were random and independent from job to job. In reality, there is time dependence between the hold intervals due to the scheduling algorithm, and the control law is hence only sub-optimal.

The predictor (7) must also be modified to work with statistics rather than known-ahead values. The state can be predicted using expected value calculations as

$$\hat{\mathbf{x}}(t_n) = \mathbb{E}\left\{A(\sigma_n)\right\} \mathbf{x}(t_n - \sigma_n) + \mathbb{E}\left\{A(\psi_{1n})B(\psi_{2n})\right\} \mathbf{u}(t_{n-2}) + \mathbb{E}\left\{B(\psi_{1n})\right\} \mathbf{u}(t_{n-1}). \tag{10}$$

Again, the predictor will only be sub-optimal due to the time-dependence induced by the scheduling algorithm.

5.3 Controller Synthesis Example

The synthesis methods presented above are illustrated in a simple control example, which was used to evaluate the performance of a standard (non-deadline-miss-aware) controller under various overrun strategies in [14]. The plant to be controlled is an integrator process described by the parameters $A_c = 0$, $B_c = 1$, $Q_{1c} = 1$, $Q_{2c} = 0.1$ and $R_c = 1$. The plant is controlled by a control task with stochastic execution times, executing alone in a CPU. The execution time may assume value equal to 1 s with probability 0.8, or uniformly distributed in the interval $(1, 2]$ with combined probability 0.2. For periods ranging between 1 and 2, we compare the resulting performance under the Kill, Skip-Next, and Queue(1) strategies in Figure 6. Since J_{ctl} is defined as a cost, lower values in the graph mean better performance. For each configuration, a standard controller (designed assuming no missed deadlines), a robust controller, and a clairvoyant controller are designed, and the performance of each controller, measured in terms of the cost function (2), is evaluated using JitterTime [13] in a simulation of 100,000 jobs. It can be noted that there is a strict ordering from the worst performance under standard control to the best performance under clairvoyant control, as expected. This means that designing control strategies that take into account deadline misses is beneficial in all cases. The DMAC design does not achieve the optimal cost that the clairvoyant design is able to achieve, but systematically beats classical control design due to its delay and hold compensation.

As the period is decreased from 2 to lower values, the Kill and Queue(1) strategies initially behave similarly, with decreasing cost. In fact, in the case of a miss followed by a

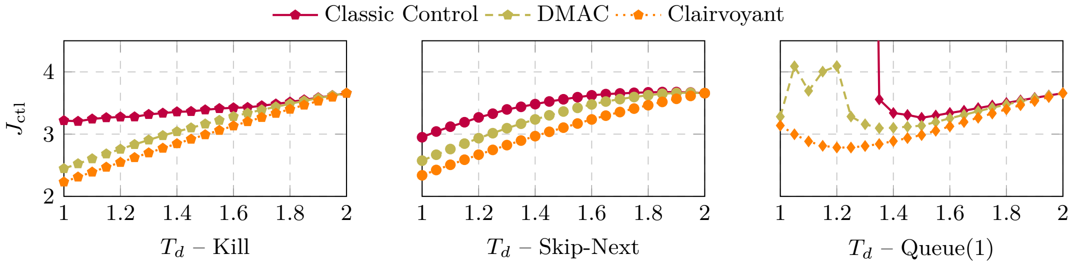

Figure 6 Control synthesis example: single task with deadline misses.

deadline hit, the Kill and Queue(1) strategies have the same behavior (since the output of the late-completed job under Queue(1) is overwritten by the completion of the next one). Skip-Next initially has an increase in cost due to the waste of resources when a very small overrun leads to a whole period being skipped. For smaller task periods, Queue(1) suffers performance degradation and even instability ($J_{\mathrm{ctl}} \to \infty$) due to the lag introduced by the queuing. The Kill and Skip-Next strategies perform the best at $T_d = 1$, with very similar results for this example.

It should be noted that the results are problem dependent, and it is hard to judge whether Kill or Skip-Next works the best in general. In all examples, however, we have found that better performance can be achieved by shortening the period and allowing a few deadline misses. Some tests that include higher-priority tasks Γ' are presented later in Section 7.

6 Stochastic Analysis

Section 5 introduced a control design technique that exploits information about the probability of sequences of deadline hits and misses for the control job. Here, we provide a framework to robustly estimate these probabilities, and at the same time preserve a pessimistic bound that allows us to mitigate the effect of worst-case conditions. We formulate the estimation problem as a chance-constrained optimization problem [37], i.e., an optimization problem where we look for the probabilities of different sequences of hits and misses given the worst-case realization of the uncertainty inherently present in the taskset execution.

Analytical approaches extracting the probability of hits and misses for a schedule of jobs are either extremely pessimistic [17] or have a high computational complexity [51]. This limits the applicability of these techniques in non-trivial cases. Moreover, there are few works dealing with joint probabilities of consecutive jobs, like [49], but they still lack of scalability.

To handle the scalability issue, we adopt a simulation-based approach, backed up by the *scenario theory* [11], that *empirically* performs the uncertainty characterization, and provides *formal guarantees* on the robustness of the resulting estimation. The scenario theory allows us to exploit the fact that simulating the taskset execution (with statistical significance) is less computationally expensive than an analytical approach that incurs into the problem of combinatorial explosion of the different possible uncertainty realizations. In practice, this means that we: (i) sample the execution times from the probability distributions specified for each task, $f_i^{\mathcal{C}}(c)$, (ii) schedule the tasks, checking the resulting set of sequences Ω, and (iii) find the worst-case sequence ω_* based on the chosen cost function. The probabilities of sequences of hits and misses are then computed based on this sequence, and used in the design of the controller to be robust with respect to the sequence. We use the scenario theory to quantify, according to the number of extracted samples, the probability ε of not having extracted the *true* worst-case sequence and the confidence in the process $1 - \beta$. Scenario theory has for example found use in the management of energy storage [20].

6.1 Scenario Theory

The scenario theory has been developed in the field of robust control to provide robustness guarantees for convex optimization problems in presence of probabilistic uncertainty. In these problems, accounting for all the possible uncertainty realization might be achieved analytically, but is computationally too heavy or results in pessimistic bounds. The scenario theory proposes an empirical method in which samples are drawn from the possible realizations of uncertainty, finding a lower bound on the number of samples. It provides statistical guarantees on the value of the cost function with respect to the general case, provided that the sources of uncertainty are the same.

One of the advantages of this approach is that there is no need to enumerate the uncertainty sources, the only requirement being the possibility to draw representative samples. This eliminates the need to make assumptions on the correlation between the probability of deadline misses in subsequent jobs. If interference is happening between the jobs, this interference empirically appears when the system behavior is sampled. While there is no requirement on subsequent jobs interfering with one another, there is a requirement that different sequences are independent (i.e., each sequence represents an execution of the entire taskset of a given length, in the same or possibly different conditions). Taking the worst observed case in a set of experiments, the scenario theory allows us to estimate the probability that something worse than what is observed can happen during the execution of the system.

Specifically, for a sequence ω we define a cost function $J_{seq}(\omega)$, that determines when we consider a sequence worse than another (from the perspective of the controller execution). Denoting with $\mu_{\text{tot}}(\omega)$ the total number of job skips and deadline misses that the control task experienced in ω, and with $\mu_{\text{seq}}(\omega)$ the maximum number of consecutive deadline misses or skipped jobs in ω, we chose to use as a cost function the following expression:

$$J_{seq}(\omega) = \mu_{\text{tot}}(\omega)\, \mu_{\text{seq}}(\omega) \tag{11}$$

to determine the worst-case sequence of hits and misses. Given a set of sequences $\Omega = \{\omega_1, \dots \omega_{n_{\text{sim}}}\}$, we select $\omega_* = \arg\max_{\omega \in \Omega} J_{seq}(\omega)$. The choice of the cost function is anyhow not-univocal. For instance, other viable alternatives would be: (i) the number of subsequences of a given length with at least a given number of deadline misses, or (ii) the shortest subsequence with more than a given number of deadline misses.

6.2 Formal Guarantees

The scenario theory allows us to compute the number n_{sim} of simulations that we need to conduct to reach the required robustness ε and confidence $1 - \beta$. The parameter ε is a bound on the probability of the obtained result being wrong, i.e., on the probability that another simulation would lead to a sequence with a higher cost function J_{seq} than ω_*. The parameter $1 - \beta$ represents the confidence we have in this result, i.e., the probability of ε being an incorrect bound. It can also be interpreted as the probability that the drawn n_{seq} sequences are representative enough of the whole set of possible uncertainty realizations.

Equation (12) shows the relation between the number of experiments n_{sim}, ε and β [11]. Here, d is the number of optimization variables used for the selection. The cost function J_{seq}

that we defined takes as argument only a sequence ω, hence $d = 1$.

$$\sum_{i=0}^{d-1} \binom{n_{\text{sim}}}{i} \varepsilon^i (1-\varepsilon)^{n_{\text{sim}}-i} \leq \beta. \tag{12}$$

Specifying β and ε univocally determines n_{sim}. If β and ε are sufficiently small, we can use the worst-case sequence for the design of the controller with high confidence.

6.3 Application and Threats to Validity

Similarly to any other empirical approach, the validity of the scenario theory depends on the representativeness of the sampling set. In our case, for example the validity of our results depends on the significance of the probabilistic execution time distributions for all the tasks.

Furthermore, the length of the simulations is a critical parameter. We simulate the system for a number n_{job} of executions of the control task. Clearly, we want to select n_{job} to cover an entire hyperperiod (to achieve complete analysis of the interferences between the tasks). In practice, we want to be able to detect cascaded effects that might happen due to the probabilistic nature of the execution times of the tasks. Some samplings could in fact make the utilization of instances of the taskset greater than one. For this reason simulations that include several hyperperiods should be performed. On top of that significancy with respect the controlled of the physical system is required (since the existence of the hyperperiod is not always guaranteed), hence the length of the simulated sequences should cover its dynamics.

7 Experimental Evaluation

This section presents and discusses the results obtained with our synthesis method. Experiments are obtained generating synthetic real-time workload. First, we generate the tasks Γ' and τ_d. We then use a simulator to draw execution time realizations from the determined probability distributions for the tasks and generate schedules and sequences of deadline hits and miss with different deadline-miss handling strategies, according to the scenario theory parameters. We select the worst-case sequence and use it for control design. Finally, we test the obtained controller on the physical plant, computing the control performance J_{ctl}. Section 7.1 describes our experimental setup, while Section 7.2 discusses our results.

7.1 Setup

To generate the taskset, and its execution time probability distributions, our experimental evaluation follows this procedure:

- Using the UUnifast algorithm [8], we generate an initial taskset Γ', composed of $N_T - 1$ tasks and having utilization $U_{\Gamma'}$. We order the tasks using Rate Monotonic priority. In the following, we show examples where $N_T \in \{5, 10, 20\}$, and $U_{\Gamma'} \in \{0.70, 0.80\}$. Tasks periods are chosen randomly from a bucket of values, ranging between 100 ms and 1000 ms, with steps of 10 ms. The execution times generated by the UUnifast algorithm are set as the WCETs of the tasks. All tasks in the generated task set must respect their (hard) deadlines using the WCET values.
- For each generated taskset, we build a control task τ_d. The control task is set to have the lowest priority in the set. We assume that the interval of interesting periods for the controller spans between 0.5 s and 2 s. These values are chosen according to the physical constraints that the plant imposes (e.g., the speed of the plant dynamics). We then select a random value for the WCET of τ_d, ensuring that the response time of its critical job

(which corresponds to the WCRT of τ_d in case no deadline is missed) is between 2 s and 2.5 s. This choice guarantees that in the interval of interest, the control task has non-zero probability of missing at least one deadline.

- For each task in the taskset we randomly choose the BCET, such that the the BCRT of τ_d lies below the lower limit of 0.5 s of our interval of interest (coherently with our hypothesis that the controller period should be higher than its BCRT). Since the execution time probability of each task is skewed towards lower values, we expect that the probability distribution of response times will be skewed in the same direction too. We experimented with many values for the controller BCRT R_d^B. We found two representative intervals, that show different trends and behaviors and therefore selected for visualization the cases in which $R_d^B \in [0.15, 0.25]$ s and $R_d^B \in [0.4, 0.5]$ s.
- For each task, we choose N_e points uniformly spaced in the interval between the task BCET and its WCET as possible execution times. In our tests, we selected $N_e = 5$, but a higher number of points does not pose any scalability issue. The probabilities for each possible execution time have been assigned with the following heuristic: the lowest half (rounded up) of values have assigned a probability of that cumulatively sums up to 0.75, while the remaining sums up to $1 - 0.75 = 0.25$. This choice is based on the assumption that, realistically, the probabilities should be *skewed* towards the lower values.

For each period of interest T_d, we evaluate the control design as follows. We use a simulator, built in C++, to generate the sequences used for the scenario theory. We choose $\epsilon = 0.003$ and $\beta = 0.01$, obtaining a value for the number of simulations equal $n_{\text{sim}} = 1533$. A number $n_{\text{j}} = 500$ of control jobs has been chosen as representative temporal horizon for our system. After choosing a target period T_d for the controller in the interval $[0.5, 2]$ s, the simulator generates a vector of jobs activated in the time interval $n_{\text{job}}\, T_d$. Each job is characterized by its activation instant, priority, deadline, and an execution time (drawn from the probability distribution described above). The simulator computes the response time and the delay σ_k of each control job, by using the three deadline miss strategies – i.e., Kill, Skip-Next, Queue(1). For each sequence of control jobs the simulator computes a cost J_{seq}, weighting both the total number of deadline misses and the maximum number of consecutive ones as shown in Equation (11).

The worst-case sequence is the input of our control design script, expressed as a vector of delays. The Matlab control design script computes the hold interval using the rules presented in Section 4.1, and selects the set of valid control jobs from the sequence. The average probability of all combinations of delay and hold values are extracted from the sequence, and used to build the DMAC controller, as shown in Section 5.2.

Finally, the performance of the controlled system, where the control update is driven by the sequence of delays and holds from task schedule, is computed using *JitterTime* [13], a simulation-based tool for analysis of control systems performance inspired by *Jitterbug* [34] and *TrueTime* [15]. This new tool, built entirely in Matlab, is able to model transitions between different states with variable and conditional probabilities (overcoming some of the limitations of Jitterbug). JitterTime tests the various sequences of job schedules (with randomly generated execution times), producing the average and worst-case performance of the controller in terms of the cost function J_{ctl}.

7.2 Results

We conduct some experiments to study how the control performance, defined in Equation (2), changes when the control task period T_d assumes different values in the interval of interest.

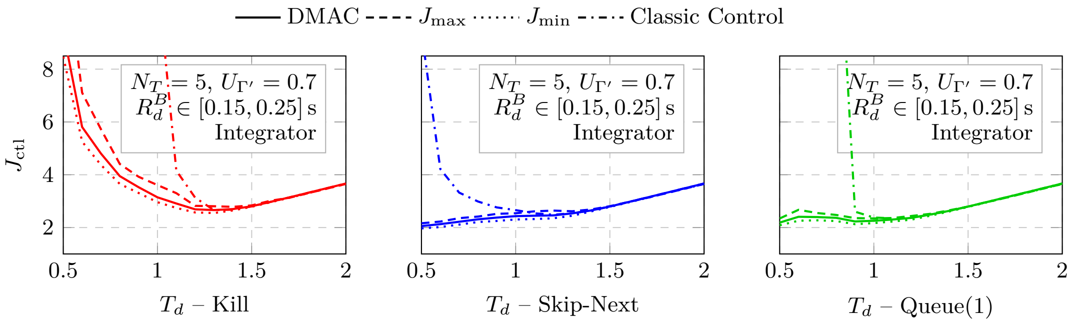

Figure 7 Comparison of DMAC with different deadline miss handling strategies (average, maximum, and minimum performance) with classical control.

In the following, we show results obtained in different configurations. Specifically, for each deadline miss strategy we vary: (i) the number N_T of tasks in the taskset, (ii) the utilization of Γ', (iii) the control task best case response time R_d^B, and (iv) the dynamics of the physical plant to be controlled.

Figure 7 shows that DMAC design outperforms the classical control design (remember that J_{ctl} is defined as a cost, thus the lower, the better). We use the same plant as described in Section 5.3. From left to right, the figures show the cost function J_{ctl} obtained with the Kill, the Skip-Next, and the Queue(1) strategy when the period T_d varies. Solid lines represent the average performance of the DMAC controller (in the n_{sim} simulations). Dash dotted lines show the average performance of the classical control design method. When the period decreases, DMAC consistently and increasingly outperforms the classical design, obtaining a lower cost function. To ensure the robustness of the DMAC controller, we also plot the maximum and minimum cost obtained during the n_{sim} simulations (respectively using dashed and dotted lines). The area between the maximum and the minimum cost (which apparently includes the average value) is narrow, validating the robustness claim.

In Figure 8 we investigate the effect of varying the number of tasks N_T and the best case response time of the control task R_d^B. The number of tasks does not have a dramatic effect on any of the controllers, but the performance of a controller with the Kill strategy seem to benefit from an increase in the number of tasks. More generally, however, the Kill strategy is dominated by both the Skip-Next and the Queue(1) performance, that allow the design to reach shorter periods and to lower the cost function. The Kill strategy, that was achieving very good performance when tested with a single control task, does not handle additional load well. In fact, the failure of the Kill strategy is due to cascaded effects – killing subsequent jobs due to interference introduces long delays for the control signal, while allowing the job to terminate anyway (as the other two strategies do) leads to better performance. The Queue(1) and Skip-Next strategies behave similarly for both low values of R_d^B and high number of tasks. However, the Queue(1) shows some local performance drop in the cases of higher R_d^B. This happens when the average delay is just before the threshold of $2T_d$, i.e. when high delays but few skips occur. We then conclude that Skip-Next strategy is the most robust to both variations in the number of tasks N_T and in the best case response time R_d^B.

We conducted an extensive amount of tests, but due to space restrictions we can only show a limited number of additional results. Figure 9 shows the effect of increasing the utilization of Γ' and of changing the plant dynamics. An increase in utilization does not change the trends that can be observed (the Kill strategy being outperformed). However, changing the dynamics of the plant from a (marginally stable) Integrator to an (unstable)

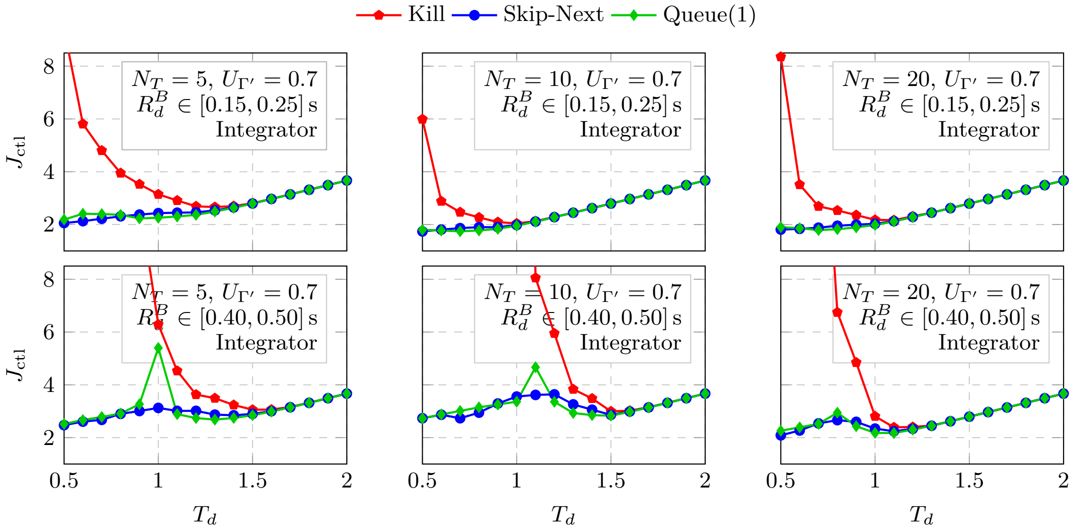

Figure 8 Control cost function for the Integrator plant with different miss-handling strategies, varying the number of tasks, and the best case response time.

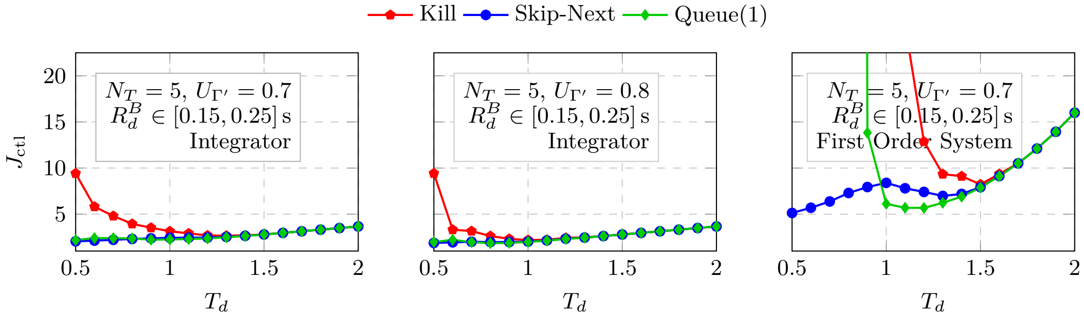

Figure 9 Control cost function varying utilization and plant dynamics.

First-Order system has a dramatic effect on the cost function. The Queue(1) strategy behaves similarly (although better) with respect to the Kill strategy, not being able to handle short periods. The Skip-Next strategy, on the contrary, shows robust performance with respect to period shortening. The qualitative dependence on the plant dynamics will be explored further in future research.

8 Related Work

When designing a discrete time controller, it is fundamental to study if timing non-idealities may occur and how much they could harm the performance of the controlled system. The problem of analyzing the effects of late information on the system performance [31] has raised particular interest, especially in networked systems. In fact, transmission delays and packet drops may happen frequently when the transmission channels are heavily loaded or noisy. These timing effects are usually characterized as independent events with Gaussian distributions, or using worst case bounds [6]. By leveraging the knowledge of the timing non-idealities, many works proposed solutions for assuring the stability of the system [10, 35] and improving the control performance [44]. Sinopoli et al. [45] proposed an optimal control design for networked system leveraging the probability of packet losses. Similarly, the problem of designing an optimal control considering packet drops from the sensor is faced in [26] and [50]. In [46], the authors design an adaptive control that switches between normal, abort and skip mode depending on the delay (but which is always lower than than the period).

When dealing with controllers implemented in real-time systems, however, a different and more complex analysis is needed. Here, the input-output delay experienced by the control flow comes from the interference of higher priority tasks due to limited computational resources, that may even cause some job to miss their deadlines. Unforeseen delays may be caused, for example, by overload activations [27, 54], cache misses [3, 22] or complex interactions between scheduling and system state [9]. In recent works, systems that experience deadline misses are described using the so called weakly-hard model [7]. In this model, the possibility of missing a deadline is upper-bounded by a constraint (m, K), which gives the maximum number of deadlines m that may happen every K activation of a task. This model has proved being suitable for studying the effects of missed deadlines on the performance of control tasks and scheduling [24, 42]. A detailed modeling of the control performance considering different deadline miss handling strategies is presented in [41]. The effects of missed deadlines on system performance have been studied also using co-simulation [40]. Other works faced the co-design problem in overloaded systems by using complex mechanisms that take into account system stability and processor load [19, 25, 56].

In this paper, we study the effects of missed deadlines on the control performance by describing miss and hit events in a probabilistic fashion. The urge to bound WCET estimation and ensure timing correctness of systems led to the development of many probabilistic modeling techniques with remarkable success when applied to real systems [12, 21, 30, 52].

In our paper we assume that the execution times of each job are assumed as independent [2, 36], but we overcome the limitation of classical approach, not requiring that response times are modeled as independent variables. Exact methods for computing probabilistic response times of jobs exist [23], but their major downside is that they do not scale well and are applicable only to limited task sets and short hyperperiods. Other approaches face the problem by extracting probabilistic bounds with various approximation techniques [17, 18, 51]. The particular case of extracting joint probabilities of successive jobs is however less studied, and has been found e.g. in [49]. Again the work in [17] develops a bound for l-consecutive deadline misses, but it is still insufficient for our purposes. The path chosen for our work leverages the *scenario theory* approach [11] for performing a robust estimate of the hit and miss probabilities, by simulating multiple possible schedules.

9 Conclusion

This paper presented DMAC, a novel control design technique for building a fixed *Deadline-Miss-Aware Controller*. It also contributes with a methodology to evaluate control design strategies in the presence of deadline misses and possible overruns. Our controller leverages the probability of possible sequences of missed deadlines to compensate for the introduced delays. Our experimental results show that our design obtains better performance, while (safely) exploring period ranges that are usually avoided in state-of-the-art approaches. Moreover, we discussed how the control performance changes with respect to different deadline miss strategies and different taskset parameters. This paper highlights how, choosing the deadline miss handling strategy is one of the most critical parameters in the control design.

References

1 Leonie Ahrendts, Sophie Quinton, and Rolf Ernst. Finite ready queues as a mean for overload reduction in weakly-hard real-time systems. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, pages 88–97. ACM, 2017.

2 Sebastian Altmeyer, Liliana Cucu-Grosjean, and Robert I Davis. Static probabilistic timing analysis for real-time systems using random replacement caches. *Real-Time Systems*, 51(1):77–123, 2015.

3 Sebastian Altmeyer and Robert I Davis. On the correctness, optimality and precision of static probabilistic timing analysis. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 26. European Design and Automation Association, 2014.

4 Amir Aminifar, Soheil Samii, Petru Eles, Zebo Peng, and Anton Cervin. Designing high-quality embedded control systems with guaranteed stability. In *2012 IEEE 33rd Real-Time Systems Symposium*, pages 283–292. IEEE, 2012.

5 Karl J Åström and Björn Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.

6 Philip Axer, Maurice Sebastian, and Rolf Ernst. Probabilistic response time bound for CAN messages with arbitrary deadlines. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*, pages 1114–1117. IEEE, 2012.

7 Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE transactions on Computers*, 50(4):308–321, 2001.

8 Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2):129–154, 2005.

9 Alessandro Biondi, Marco Di Natale, Giorgio C Buttazzo, and Paolo Pazzaglia. Selecting the transition speeds of engine control tasks to optimize the performance. *ACM Transactions on Cyber-Physical Systems*, 2(1):1, 2018.

10 Rainer Blind and Frank Allgöwer. Towards networked control systems with guaranteed stability: Using weakly hard real-time constraints to model the loss process. In *Decision and Control (CDC), 2015 IEEE 54th Annual Conference on*, pages 7510–7515. IEEE, 2015.

11 Giuseppe C Calafiore and Marco C Campi. The scenario approach to robust control design. *IEEE Transactions on Automatic Control*, 51(5):742–753, 2006.

12 Francisco J Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, et al. Proartis: Probabilistically analyzable real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 12(2s):94, 2013.

13 Anton Cervin. JitterTime 1.0 reference manual. Technical Report TFRT-7658, Department of Automatic Control, Lund University, 2019.

14 Anton Cervin. Analysis of overrun strategies in periodic control tasks. In *Proc. 16th IFAC World Congress, Prague, Czech Republic*, page 137. Citeseer, 2005.

15 Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and K-E Arzen. How does control timing affect performance? analysis and simulation of timing using Jitterbug and TrueTime. *IEEE control systems*, 23(3):16–30, 2003.

16 Anton Cervin, Bo Lincoln, Johan Eker, Karl-Erik Arzén, and Giorgio Buttazzo. The jitter margin and its application in the design of real-time control systems. In *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, pages 1–9. Gothenburg, Sweden, 2004.

17 Kuan-Hsun Chen and Jian-Jia Chen. Probabilistic schedulability tests for uniprocessor fixed-priority scheduling under soft errors. In *Industrial Embedded Systems (SIES), 2017 12th IEEE International Symposium on*, pages 1–8. IEEE, 2017.

18 Kuan-Hsun Chen, Georg Von Der Brüggen, and Jian-Jia Chen. Analysis of deadline miss rates for uniprocessor fixed-priority scheduling. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 168–178. IEEE, 2018.

19 Hoon Sung Chwa, Kang G Shin, and Jinkyu Lee. Closing the gap between stability and schedulability: a new task model for cyber-physical systems. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 327–337. IEEE, 2018.

20 G. Darivianakis, A. Eichler, R. S. Smith, and J. Lygeros. A data-driven stochastic optimization approach to the seasonal storage energy management. *IEEE Control Systems Letters*, 1(2):394–399, 2017.

21 Robert Davis, Tullio Vardanega, Franck Wartel, Liliana Cucu-Grosjean, et al. PROXIMA: a probabilistic approach to the timing behaviour of mixed-criticality systems. *Ada User Journal*, 2:118–122, 2014.

22 Robert I Davis, Luca Santinelli, Sebastian Altmeyer, Claire Maiza, and Liliana Cucu-Grosjean. Analysis of probabilistic cache related pre-emption delays. In *Real-Time Systems (ECRTS), 2013 25th Euromicro Conference on*, pages 168–179. IEEE, 2013.

23 José Luis Díaz, Daniel F García, Kanghee Kim, Chang-Gun Lee, L Lo Bello, José María López, Sang Lyul Min, and Orazio Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 289–300. IEEE, 2002.

24 Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *RTSS*, pages 53–62, 2014.

25 Mongi Ben Gaid, Daniel Simon, and Olivier Sename. A design methodology for weakly-hard real-time control. *IFAC Proceedings Volumes*, 41(2):10258–10264, 2008.

26 Vijay Gupta, Babak Hassibi, and Richard M Murray. Optimal LQG control across packet-dropping links. *Systems & Control Letters*, 56(6):439–446, 2007.

27 Zain AH Hammadeh, Sophie Quinton, and Rolf Ernst. Extending typical worst-case analysis using response-time dependencies to bound deadline misses. In *Embedded Software (EMSOFT), 2014 International Conference on*, pages 1–10. IEEE, 2014.

28 Thomas A Henzinger, Benjamin Horowitz, and Christoph Meyer Kirsch. Giotto: A time-triggered language for embedded programming. In *International Workshop on Embedded Software*, pages 166–184. Springer, 2001.

29 Byung Kook Kim. Task scheduling with feedback latency for real-time control systems. In *Real-Time Computing Systems and Applications, 1998. Proceedings. Fifth International Conference on*, pages 37–41. IEEE, 1998.

30 Leonidas Kosmidis, Jaume Abella, Eduardo Quiñones, and Francisco J Cazorla. A cache design for probabilistically analysable real-time systems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 513–518. EDA Consortium, 2013.

31 Antzela Kosta, Nikolaos Pappas, Anthony Ephremides, and Vangelis Angelakis. Age and value of information: Non-linear age case. In *Information Theory (ISIT), 2017 IEEE International Symposium on*, pages 326–330. IEEE, 2017.

32 Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

33 John P Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209. IEEE, 1990.

34 Bo Lincoln and Anton Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, volume 2, pages 1319–1324. IEEE, 2002.

35 Steffen Linsenmayer and Frank Allgower. Stabilization of networked control systems with weakly hard real-time dropout description. In *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*, pages 4765–4770. IEEE, 2017.

36 Rui Liu, Alex F Mills, and James H Anderson. Independence thresholds: Balancing tractability and practicality in soft real-time stochastic analysis. In *Real-Time Systems Symposium (RTSS), 2014 IEEE*, pages 314–323. IEEE, 2014.

37 Bruce L. Miller and Harvey M. Wagner. Chance constrained programming with joint constraints. *Oper. Res.*, 13(6), 1965.

38 Johan Nilsson, Bo Bernhardsson, and Bjorn Wittenmark. Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1), 1998.

39 Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The ROSACE case study: From Simulink specification to multi/many-core execution. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 309–318, 2014.

40 Paolo Pazzaglia, Marco Di Natale, Giorgio Buttazzo, and Matteo Secchiari. A framework for the co-simulation of engine controls and task scheduling. In *International Conference on Software Engineering and Formal Methods*, pages 438–452. Springer, 2017.

41 Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. Beyond the weakly hard model: Measuring the performance cost of deadline misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

42 Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):549–559, 1999.

43 Ola Redell and Martin Sanfridson. Exact best-case response time analysis of fixed priority scheduled tasks. In *Real-Time Systems, 2002. Proceedings. 14th Euromicro Conference on*, pages 165–172. IEEE, 2002.

44 Luca Schenato, Bruno Sinopoli, Massimo Franceschetti, Kameshwar Poolla, and S Shankar Sastry. Foundations of control and estimation over lossy networks. *Proceedings of the IEEE*, 95(1):163–187, 2007.

45 Bruno Sinopoli, Luca Schenato, Massimo Franceschetti, Kameshwar Poolla, and Shankar Sastry. An lqg optimal linear controller for control systems with packet losses. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 458–463. IEEE, 2005.

46 Damoon Soudbakhsh, Linh Thi Xuan Phan, Anuradha M Annaswamy, and Oleg Sokolsky. Co-design of arbitrated network control systems with overrun strategies. *IEEE Transactions on Control of Network Systems*, 5(1):128–141, 2018.

47 Youcheng Sun and Marco Di Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):171, 2017.

48 Zhendong Sun. *Switched linear systems: control and design*. Springer Science & Business Media, 2006.

49 Bogdan Tanasa, Unmesh D Bordoloi, Petru Eles, and Zebo Peng. Probabilistic response time and joint analysis of periodic tasks. In *Real-Time Systems (ECRTS), 2015 27th Euromicro Conference on*, pages 235–246. IEEE, 2015.

50 Eelco P van Horssen, AR Baghban Behrouzian, Dip Goswami, Duarte Antunes, Twan Basten, and WPMH Heemels. Performance analysis and controller improvement for linear systems with (m, k)-firm data losses. In *2016 European Control Conference (ECC)*, pages 2571–2577. IEEE, 2016.

51 Georg von der Brüggen, Nico Piatkowski, Kuan-Hsun Chen, Jian-Jia Chen, and Katharina Morik. Efficiently approximating the probability of deadline misses in real-time systems. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 106 of *ECRTS*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

52 Franck Wartel, Leonidas Kosmidis, Code Lo, Benoit Triquet, Eduardo Quinones, Jaume Abella, Adriana Gogonel, Andrea Baldovin, Enrico Mezzetti, Liliana Cucu, et al. Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study. In *Industrial Embedded Systems (SIES), 2013 8th IEEE International Symposium on*, pages 241–248. IEEE, 2013.

53 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, et al.

The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(3):36, 2008.

54 Wenbo Xu, Zain AH Hammadeh, Alexander Kroller, Rolf Ernst, and Sophie Quinton. Improved deadline miss models for real-time systems using typical worst-case analysis. In *2015 27th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 247–256. IEEE, 2015.

55 Yang Xu, Karl-Erik Årzén, Anton Cervin, Enrico Bini, and Bogdan Tanasa. Exploiting job response-time information in the co-design of real-time control systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on*, pages 247–256. IEEE, 2015.

56 Tatsuya Yoshimoto and Toshimitsu Ushio. Optimal arbitration of control tasks by job skipping in cyber-physical systems. In *Proceedings of the 2011 IEEE/ACM Second International Conference on Cyber-Physical Systems*, pages 55–64. IEEE Computer Society, 2011.