Stochastic Stability Analysis of Control Systems Subject to Communication and Computation Faults

Authors omitted for double-blind submission – RTAS 2023 Submission #62, 11 pages

***Abstract*—Control theory allows one to design controllers that are robust to external disturbances, model simplification, and modelling inaccuracy. Researchers have investigated whether the robustness carries on to the controller's digital implementation, mostly looking at how the controller reacts to either communication or computational problems. Communication problems are typically modelled using random variables (i.e., estimating the probability that a fault will occur during a transmission), while computational problems are modelled using deterministic guarantees on the number of deadlines that the control task has to meet. These fault models allows the engineer to both design robust controllers and assess the controllers' behaviour in the presence of faults. However, the question of what happens when these faults occur simultaneously, as is the case for real implementations, is still open. In this paper, we answer this question in the stochastic setting, using the theory of Markov Jump Linear Systems to provide stability contracts with *almost sure* guarantees of convergence. We apply our method to three case studies from the recent literature and show their robustness to a comprehensive set of faults.**

I. Introduction

Real-time control systems are, by design, robust to disturbances and modelling inaccuracy [32]. Their robustness typically carries on to sporadic implementation problems, like communication dropouts and computational delays. Despite this, the first digital controllers were implemented as hard real-time periodic tasks, following the classical Liu and Layland model [24]. Due to the inherent robustness of controllers, this model was soon considered overly conservative. In particular, safe and conservative real-time analysis led to an increase in the controller's sampling period, hence reducing the control performance. To correct the mismatch between performance and robustness, researchers adopted either the soft real-time task model [5] or the weakly-hard task model [2].

The weakly-hard model was originally devised to provide formal guarantees to tasks that can tolerate occasional deadline overruns, e.g., controller tasks where decreasing the sampling period improves the control performance whilst introducing deadline overruns. The clear, predictable, and bounded mechanisms that the weakly-hard model provide were quickly adopted by the research community, generating results in (among many others) networked communication [1], [23] and control [26], [31]. The main appeal with the weakly-hard model for control systems is the ability to guarantee properties of the closed-loop systems in the worst-case conditions. However, if the worst-case is rare, the general results may be exceedingly conservative under normal operation [40]. Furthermore, despite the existence of techniques to extract weakly-hard guarantees from systems, e.g., [37], it is still fairly uncommon to obtain a weakly-hard characterisation of a hardware and software platform.

On the contrary, typical fault models are probabilistic. If some risk is tolerated, or if the worst-case is extremely rare, *soft* (*probabilistic* or *stochastic*) models can significantly improve typical-case performance; in particular since the probabilistic models aim to optimise the average-case performance rather than the worst-case robustness. There exists a vast literature on stochastic stability for control systems [3], [4], [13], [20], [21]. However, these results are typically not applied to computational problems, and usually do not consider that multiple faults can occur simultaneously, i.e., there might be a channel dropout in addition to the controller code stalling and not completing its execution before its deadline.

There seems to be a misconception that packet dropouts in networked control system can be used to analyse control deadline overruns in the hardware, and vice versa. In this paper we aim to resolve the confusion and provide a unified analysis that handles *simultaneously*: (i) packet losses on the sensor channel, (ii) computational overruns of the control task, and (iii) packet losses on the actuator channel. Specifically:

- We compile a model of what happens in the control system when the different faults are experienced. This model includes both the discrete state (which encodes whether message transmissions and control computation have been successful) and the dynamical state of the system (which encodes the physical quantities that are affected by the controller execution).
- We leverage the literature on stochastic control, and in particular the modelling framework of Markov Jump Linear Systems [10], to provide a stochastic analysis of control systems subject to simultaneous communication and computation faults.
- We apply the analysis to three different case studies taken from the literature, showing how resilient their controllers are to faults that may occur during the lifetime of the controller.

The rest of this paper is outlined as follows. Section II contains the necessary background and a clear problem statement. In Section III we propose an analysis of control systems subject to *both* packet dropouts and computational overruns. In Section IV we evaluate the analysis on three different case studies taken from the literature, and show that we can assess the (stochastic) stability of the controlled systems under a variety of simultaneous faults. Section V summarises the related literature and Section VI concludes the paper.

II. Problem Formulation

This section provides the necessary background and introduces the cyber-physical system models used in the remainder of the paper. In particular, Section II-A provides a brief overview of the models used in the design of linear control systems and Section II-B discusses the implementation choices made in the realisation of the controller code. Finally, Section II-C formalises the problem addressed in this paper.

A. Control System Design

The objective of a feedback control system is to make a physical process (denoted *plant*) behave according to some predetermined requirements. Such requirements generally include stabilising the plant, rejecting disturbances, and tracking a desired trajectory. Stability is essential to guarantee that physical quantities, like the velocity of a vehicle, stay bounded.

In their most general form, the plant dynamics are continuous-time and non-linear. However, for control analysis and synthesis purposes [32], a simpler model of the plant is generally devised and discretised, generally obtaining a discrete-time Linear Time-Invariant (LTI) state-space system. The system typically takes the form:

$$\mathcal{P}: \begin{cases} x_{k+1} &= A\,x_k + B\,u_k \\ y_k &= C\,x_k + D\,u_k. \end{cases} \tag{1}$$

In the equation, k counts the discrete number of samples elapsed since system startup, $x_k \in \mathbb{R}^{n_x}$ is the state vector representing the physical properties of the plant, $u_k \in \mathbb{R}^{n_u}$ is the control signal used to affect the plant, and $y_k \in \mathbb{R}^{n_y}$ is the sampled plant output (measurement signal) that the sensors return. The plant dynamics is encoded in the matrices $A \in \mathbb{R}^{n_x \times n_x}$, $B \in \mathbb{R}^{n_x \times n_u}$, $C \in \mathbb{R}^{n_y \times n_x}$, and $D \in \mathbb{R}^{n_y \times n_u}$. The eigenvalues of A, determine if the system is inherently stable ($\max|\operatorname{eig}(A)| < 1$) or not.

To satisfy the control requirements, a *controller* is implemented on digital hardware, such as an electronic control unit. Generally, controllers are designed and implemented following the Logical Execution Time (LET) paradigm [17], i.e., the sensor messages are received at the beginning of the control computation and the actuator messages are sent at the end of the control period.[1] Similarly to plants, controllers are generally described using discrete-time, LTI state-space systems:

$$\mathcal{C}: \begin{cases} z_{k+1} &= F\,z_k + G\,y_k \\ u_{k+1} &= H\,z_k + K\,y_k. \end{cases} \tag{2}$$

Here, $z_k \in \mathbb{R}^{n_z}$ is the controller's internal state vector. The controller dynamics is described by the matrices $F \in \mathbb{R}^{n_z \times n_z}$, $G \in \mathbb{R}^{n_z \times n_y}$, $H \in \mathbb{R}^{n_u \times n_y}$, and $K \in \mathbb{R}^{n_u \times n_z}$. The control requirements can be guaranteed using different control synthesis methods [32], leading to different matrices F, G, H, and K.[2]

Combining the dynamical models of the plant and controller, i.e., Equations (1) and (2), we obtain the *closed-loop system* $\mathcal{G}$.

[1]The LET paradigm increases timing predictability and reduces jitter at the cost of introducing a one-step delay in the control signal, i.e., u_{k+1}.

[2]A controller is *stateless* when $n_z = 0$ and *stateful* otherwise, i.e., $n_z > 0$. Stateless controllers can always be written as $\mathcal{C}: u_{k+1} = K\,y_k$.

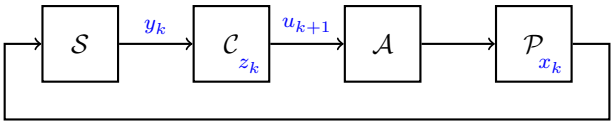

Fig. 1. Block diagram representing the interconnection of different components in a control system. The controller $\mathcal{C}$ receives input from the sensor $\mathcal{S}$ and sends data to the actuator $\mathcal{A}$, which in turn acts on the plant $\mathcal{P}$.

Similar to $\mathcal{P}$ and $\mathcal{C}$, the closed-loop system can be described by a dynamical equation:

$$\mathcal{G}: \tilde{x}_{k+1} = \Lambda\,\tilde{x}_k, \tag{3}$$

where $\tilde{x}_k$ is the closed-loop system's state vector and Λ is a matrix that encodes the closed-loop system's dynamics. For the plant and controller models used in this paper, the closed-loop state vector can be reduced down to $\tilde{x}_k = [x_k^{\mathrm{T}}, z_k^{\mathrm{T}}, u_k^{\mathrm{T}}]^{\mathrm{T}}$, where T is the transpose operator. The nominal behaviour of $\mathcal{G}$ can then be described by:

$$\underbrace{\begin{bmatrix} x_{k+1} \\ z_{k+1} \\ u_{k+1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} A & 0 & B \\ GC & F & GD \\ KC & H & KD \end{bmatrix}}_{\Lambda} \underbrace{\begin{bmatrix} x_k \\ z_k \\ u_k \end{bmatrix}}_{\tilde{x}_k} \tag{4}$$

To assess whether a closed-loop system is stable under nominal conditions or not, it is sufficient to check the eigenvalues of the closed-loop state matrix Λ. In other words, $\mathcal{G}$ is said to be Schur stable if *all* the eigenvalues of the matrix Λ lie inside the unit disc [32]. Denoting with $\sigma(\Lambda)$ the largest absolute magnitude of an eigenvalue of Λ, then $\mathcal{G}$ is said to be Schur stable if and only if the following condition holds:

$$\sigma(\Lambda) = \max|\operatorname{eig}(\Lambda)| < 1. \tag{5}$$

B. Control System Implementation

A schematic implementation of a closed-loop real-time control system $\mathcal{G}$ can be seen in Figure 1. Starting from the *sensors* $\mathcal{S}$, the plant is sampled at discrete time instants k. The controller $\mathcal{C}$ then polls the sensor channel for the latest sampled measurement signal to use in the calculation of the new control command u_k. When the new control command is computed, the controller stores it in memory before sending it to the *actuator* $\mathcal{A}$ at the beginning of the next control iteration. Finally, the actuator acts upon the plant $\mathcal{P}$, based on the command it received from the controller.

In this paper, we aim at analysing the closed-loop system in the presence of faults. We assume that three components in the real-time control system can experience faults:

(i) the *sensor channel*, that transmits information between the sensor $\mathcal{S}$ and controller $\mathcal{C}$,
(ii) the *control task* that executes the algorithm of the controller $\mathcal{C}$, and
(iii) the *actuator channel*, that allows the controller $\mathcal{C}$ to communicate with the actuator $\mathcal{A}$.

We provide a brief review of what can cause problems for both the controller and the input/output (IO) channels, together with some considerations on common implementation details.

Controller: Generally, controllers $\mathcal{C}$ are implemented as periodic tasks with implicit deadlines (denoted *control tasks*). A typical control algorithm implementation is the following.

```
while True:
  y = read_sensor_ch()
  u, z = compute_control(y, z)
  sleep_until(next_activation)
  send_actuator_ch(u)
```

Listing 1. Typical control algorithm execution.

The code in Listing 1 performs the following operations: (i) it samples the current plant measurements in `y` using the sensor $\mathcal{S}$ (via the function `read_sensor_ch`), (ii) it calculates and stores in memory the next control signal `u` and the controller's updated state `z` (via `compute_control`), (iii) it sleeps until the next activation (via `sleep_until`), and (iv) it sends the control commands to the actuators $\mathcal{A}$ (via `send_actuator_ch`).

For the control task, each iteration of the loop in Listing 1 is a new *job*, and the k-th iteration corresponds to the job j_k. The control job j_k is *released* at time $a_k = k\,p$, where p is the *period* of the control task. The objective of each job is to complete its execution before its corresponding *deadline* $d_k = a_k + p = (k+1)\,p$. We denote with f_k the time instant in which the control task *completes* the execution of job j_k. Ideally, $f_k \leq d_k$. If $f_k > d_k$, job j_k experiences an *overrun*, or a deadline miss.

Overruns can be caused by many different factors, like cache misses when the controller tries to access its stored variables [42], preemption from higher priority tasks and interrupts [36], and timeouts due to long wait times on the sensor channel [29].

Regardless of what caused it, the scheduler needs to determine how to react to the overrun. Three *deadline overrun strategies* have been considered in literature [6]: (i) *killing* the job that overran its deadline (and releasing a new one), rolling back any (possibly partial) change performed by the job that missed its deadline, thus reverting the internal task state variables to their original value, (ii) letting the job continue its execution, *skipping* the subsequent job releases, until the current one has completed its execution, or (iii) combining the two, and letting the job continue its execution but at the same time *queueing* the subsequent job executions. In the case of skip and queue, the job that continues executing operates on outdated data. On the contrary, kill allows the task to always work with fresh data, with the risk of throwing away near-completed computations. The queue strategy has been shown to create chain effects which can severely damage control systems [6], [26], hence in this paper we only consider kill and skip as viable reactions to computational overruns.

IO Channel: Control systems rely on communication interfaces between the sensors/actuators and the controller code itself (i.e., the sensor and actuator channels). Extensive research has been conducted on the control system's stability and performance when subject to sensor packet dropouts, i.e., when a message from the sensor does not arrive at its destination, like [16], [19], [22], [23]. Losing packets over the sensor channel can lead to the controller not updating the control command, using old measurement data stored in memory, or even missing job deadlines due to prolonged waiting times. In these cases, the time in between two sampling instants is time-varying, and control design strategies can be applied to optimise the control system performance [14], [35]. However, this research usually does not take into account that packet dropouts could be combined with deadline overruns.

Despite each control job starting by polling the sensor channel for the latest data (see Listing 1), what happens to the control job if no sensor message is received is unclear. Generally, if no sensor message is received within a given time limit (typically a fraction of the deadline), the function `read_sensor_ch` *times out*, and the control algorithm can perform one of the following two actions: (i) *continue* its execution, without updating the value of `y`, thus using the previous received sensor value, or (ii) *avoid* executing the remaining instructions, terminating the computation early.

The choice of continuing or avoiding is highly dependent on the system dynamics. Under the assumption that packet losses are relatively uncommon and the controller period is typically short, using continue is advantageous. In fact, if a packet is lost, it is still likely that the previous value reported by the sensor is a reasonable approximation of the physical environment. On the other hand, avoiding the execution of the remaining instructions also prevents the controller from evolving its own state `z` and the control signal `u` in a possibly unsafe direction. Listings 2 and 3 show more realistic and updated versions of Listing 1 for the continue and avoid cases.

```
while True:
  y, timeout_triggered = read_sensor_ch(timeout_seconds)
  if timeout_triggered:
    y = y_old
  u, z = compute_control(y, z)
  y_old = y
  sleep_until(next_activation)
  send_actuator_ch(u)
```

Listing 2. Control code execution when the control computation is continued if the sensor reading function `read_sensor_ch` results in a timeout.

```
while True:
  y, timeout_triggered = read_sensor_ch(timeout_seconds)
  if not timeout_triggered:
    u, z = compute_control(y, z)
  sleep_until(next_activation)
  send_actuator_ch(u)
```

Listing 3. Control code execution when the control computation is avoided if the sensor reading function `read_sensor_ch` results in a timeout.

In Listings 2 and 3, when `timeout_seconds` time units have passed without completion, the receive function `read_sensor_ch` sets the variable `timeout_triggered` to true. In Listing 2, the control algorithm continues executing its instructions using the old sensor value `y_old`, i.e., the controller's internal state `z` will be updated and a new control command `u` will be sent to the actuators. In Listing 3, the control algorithm will not be run, i.e., the internal state `z` and the control command `u` are kept constant.

Packet dropouts on the actuator channel have not been studied as thoroughly as their sensor counterpart. The reason likely comes from the fact that the actuator response is typically

hardware-dependant and difficult to detect and compensate in the control algorithm. If the actuator does not receive a new control command when it is expecting one, it defaults to a value dependent on the *actuation mode*. The actuation mode has received more attention, with different degradation or stabilising actuation policies being proposed [25]. However, the most common approaches involve either *holding* the last received control command (i.e., $u_{k+1} = u_k$) or *zeroing* the output (i.e., $u_{k+1} = 0$). Similarly to picking continue or avoid in the case of a lost actuator message, the choice of actuation mode is non-trivial and generally depend on the control system dynamic [33], [40].

C. Problem Formulation

Many different analysis frameworks have been proposed to evaluate the computational robustness to packet loss and deadline overruns of controllers [12], [14], [23], [26]. However, these analyses have two major shortcomings: (i) they are developed in isolation, and do not combine the presence of potential problems both in the computation and in the IO channels, and (ii) they rely on knowledge about the occurrence of events like deadline overruns or packet dropouts. In fact, recent works on computational overruns [23], [26] rely on the weakly-hard task model [2] to constraint the sequence of deadline overruns; and recent work like [14] are on the contrary working on the assumption that packet losses are detected and counteracted at the control level. However, typical methods for deriving bounds on both packet dropouts and deadline overruns are probabilistic; for example, estimating the probability that a specific task in a system will overrun its deadline when a particular scheduling algorithm is employed [7], [38].

This paper aims at providing a control-theoretical analysis for how the closed-loop system robustness is affected by stochastic packet losses (on sensor and actuator channels), deadline overruns, and a combination thereof. Furthermore, we want to devise an analysis method that benefit from the state-of-the-art results on fault occurrence estimation in real-time systems implementations [7], [38]. We assume to receive, as input, a set of probabilities: the probability of overruns for the control task ρ_c, the probability of sensor channel data losses ρ_s, and the probability of actuator channel data losses ρ_a. The analysis then determines – as output – a probabilistic stability notion, that allows us to validate the control system implementation in the presence of these undesirable events.

III. Analysis

Our analysis of the closed-loop system subject to IO channel dropouts and deadline overruns is based on the theory of Markov Jump Linear Systems [10]. These systems combine the dynamics of the closed-loop system and the transition probabilities of faults and errors. In Section III-A we provide some preliminary definitions and in Section III-B we derive the control system dynamics when different events and combination thereof (deadline overruns, sensor data loss, actuator data loss) occur under the different implementation policies. In Section III-C we present the Markov chain that describes the probabilistic evolution of the discrete state of the system. Additionally, we summarise and apply the Markov Jump Linear Systems theory to the closed-loop control system, obtaining a probabilistic stability analysis.

A. Event Outcomes

Equation (4) presented the dynamical model of the closed-loop system and the closed-loop state matrix Λ in nominal conditions, i.e., in the absence of faults. However, as discussed in Section II, the system dynamics are heavily impacted by whether we miss a control computation or lose a packet on either of the communication channels. To analyse the system dynamics, we define the *outcome set* Σ for the transmission on IO channels and the computation of the controller.

Definition 1 (IO Channel Outcome Sets)**.** *We denote the set of outcomes that each message on the sensor and actuator channels can experience by* $\Sigma(\mathcal{S}) = \Sigma(\mathcal{A}) = \{\mathsf{F}, \mathsf{T}\}$*:*

- F *represent a lost message, and*
- T *represent a successfully delivered message.*

Trivially, the outcome of each packet transmission is a binary event where either: (i) the packet is successfully received (T), or (ii) the packet is lost somewhere along its route (F). Since the contents of the message (e.g., measurement data from the sensors or control commands to the actuators) is irrelevant to whether the packet is lost or not, the same outcome notation is used for both $\Sigma(\mathcal{S})$ and $\Sigma(\mathcal{A})$.

Unlike the IO channels, the outcome of a control job's computation is not necessarily a binary event; instead, it is dependent on the overrun strategy employed by the scheduler. Thus, we define the outcome sets for the control task subject to either the kill or skip strategies with respect to each execution interval, i.e., *control period*.

Definition 2 (Computational Outcome Set - Kill)**.** *We denote the set of outcomes that a control job can experience in each control period when the scheduler adopts the* kill *strategy by* $\Sigma(\mathcal{C}^{\mathsf{K}}) = \{\mathsf{M}, \mathsf{H}\}$.

- M*: a job is released, but no job is completed,*
- H*: a job is both released and completed.*

Definition 3 (Computational Outcome Set - Skip)**.** *We denote the set of outcomes that a control job can experience in each control period when the scheduler adopts the* skip *strategy by* $\Sigma(\mathcal{C}^{\mathsf{S}}) = \{\mathsf{M}, \mathsf{N}, \mathsf{H}, \mathsf{R}\}$.

- M*: a job is released but no job is completed,*
- N*: no job is either released or completed,*
- H*: a job is both released and completed,*
- R*: no job is released, but a job (released in a previous period) is completed.*

For both kill and skip, each control period contains *at most* one activated and one completed job. The main difference comes from the kill strategy terminating every job that overrun its corresponding deadline, i.e., each control period contains a new control job being released and activated. Thus, the outcome set $\Sigma(\mathcal{C}^{\mathsf{K}})$ consist of only two outcomes: a job being

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ KC & H & KD & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{THT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{THF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{TMT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{TMF}}}$$

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{FHT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{FHF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{FMT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CK}}_{\mathsf{FMF}}} \tag{6}$$

completed or a job not being completed. On the other hand, the skip strategy encompasses more diverse outcomes. For instance, the outcomes M and N both encode an overrun deadline; but M represent the start of a control computation while N correspond to its continuation. Finally, R is used to identify the *late completion* of a job, i.e., a *recovery hit*. The occurrence of R and N impose constraints on the outcome ordering. To enforce only feasible sequences of control job outcomes, we introduce the following ordering rule.

Rule 1. *For a sequence of job outcomes under the skip overrun strategy, it holds that:*

- *both* M *and* H *are restricted to directly follow an* H *or* R*,*
- *an* N *can only follow an* M*, and*
- *an* R *may only directly follow an* N *or* M*.*

B. Closed-Loop System Dynamics

From Definitions 1-3, we can now fully derive the closed-loop system dynamics' evolution in time as:

$$\tilde{x}_{k+1} = \Lambda_{sca}\, \tilde{x}_k. \tag{7}$$

As for Equation (3), $\tilde{x}_k$ is the closed-loop state vector and Λ_{sca} is the closed-loop system matrix. The system dynamics does however depend on the outcome realisations, i.e., $s \in \Sigma(\mathcal{S})$, $a \in \Sigma(\mathcal{A})$, and $c \in \Sigma(\mathcal{C}^x)$ (where x is either K or S). If a fault occurs, the closed-loop system's evolution will deviate from the nominal behaviour. As an example, the closed-loop system matrix Λ_{THT} would correspond to a control job receiving the sensor message, completing its algorithm execution in the same period as it was released, and the actuators would successfully receive the new control command. Trivially, Λ_{THT} correspond to the nominal behaviour from Equation (4). Instead, if an actuator message would be lost, the control command sent to the plant would depend on the actuator mode. Therefore, the closed-loop system would evolve according to Λ_{THF}.

Note that in each control period the system experiences a new realisation of the outcome sets, i.e., the closed-loop system matrix Λ_{sca} can switch every control period. The resulting dynamics is generally called a *switching system*. Since the closed-loop dynamics is no longer consistent between periods, the Schur stability criterion presented in Equation (5) cannot be used to assess the stability of the system, and we need to devise a probabilistic approach, that will be presented in Section III-C.

As discussed in Section II, the system dynamics change with respect to (i) the overrun strategy, (ii) the actuation mode, and (iii) the choice of timeout strategy. In this paper, we include only the stability analysis for the continue case, since under the avoid implementation the control job's outcome can be treated similarly to a deadline overrun.[3] However, we provide the analysis for both actuation modes (i.e., zero or hold) and both overrun strategies (i.e., kill or skip). The difference between zero and hold can be dealt with using specific symbols in the closed-loop matrices, but the outcome set of the control jobs changes with kill or skip. Hence, we need to analyse the two cases separately.

Kill: When a fault is experienced (deadline overrun or packet dropout), the physical state of the plant x_k continue its normal evolution. However, the closed-loop will not behave according to its design specifications. In case j_k overruns its deadline, its execution is terminated and the controller's states are rolled back, i.e., $z_{k+1} = z_k$. Furthermore, when j_k overruns *or* the actuator channel experiences a dropout, the control command defaults to a value that depends on the actuation mode, i.e., $u_{k+1} = u_k$ if the actuation mode is hold, and $u_{k+1} = 0$ if the actuation mode is zero. On the contrary, if a sensor message is not received, the control algorithm computes a new control command and updates the controller's internal state, using outdated sensor measurements.

To properly describe the closed-loop behaviour under continue and kill, we define the closed-loop state vector $\tilde{x}_k^{\mathsf{CK}} = [x_k^{\mathrm{T}}, z_k^{\mathrm{T}}, u_k^{\mathrm{T}}, \hat{y}_{k-1}^{\mathrm{T}}]^{\mathrm{T}}$. The introduced auxiliary state $\hat{y}_{k-1}$ records the old sensor value, that is used if `read_sensor_ch` times out (see Listing 2).

The set of closed-loop matrices describing the system behaviour is shown in Equation (6) for the different outcome configurations. Each outcome (s, c, and a) has 2 possible configurations in the kill case. Hence, there are $2^3 = 8$ possible sca, where $s \in \Sigma(\mathcal{S})$, $c \in \Sigma(\mathcal{C}^{\mathsf{K}})$, and $a \in \Sigma(\mathcal{A})$. In Equation (6), the variable $\Delta_{\mathcal{A}}$ is either $\Delta_{\mathcal{A}} = I$ when the hold actuator mode is used or $\Delta_{\mathcal{A}} = 0$ when the zero actuator mode is adopted. The variable $\hat{y}_k$ is not updated for $s = \mathsf{F}$, i.e., $\hat{y}_k = \hat{y}_{k-1}$. Additionally, some matrices are identical (e.g., $\Lambda_{\mathsf{TMT}} = \Lambda_{\mathsf{TMF}}$ and $\Lambda_{\mathsf{FMT}} = \Lambda_{\mathsf{FMF}}$), highlighting for example that in case of kill when the computation does not complete, receiving

[3] In fact, the possible *outcome configurations* (i.e., combinations of s, c, and a) for the avoid case are included in the set of outcome configurations for continue.

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ KC & H & KD & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{THT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ GC & F & GD & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{THF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{TMT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ C & 0 & D & 0 \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{TMF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{TNT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{TNF}}}$$

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FNT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FNF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{TRT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FRT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{TRF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FRF}}}$$

$$\underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & H & 0 & K \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FHT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & F & 0 & G \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FHF}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FMT}}} \quad \underbrace{\begin{bmatrix} A & 0 & B & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & \Delta_{\mathcal{A}} & 0 \\ 0 & 0 & 0 & I \end{bmatrix}}_{\Lambda^{\mathsf{CS}}_{\mathsf{FMF}}} \tag{8}$$

or not receiving the actuator signal is irrelevant for the system evolution.

Skip: Similarly to the kill case, when the computation mode for overruns is set to skip, the physical states x_k continue their evolution. However, in contrast to the kill overrun strategy, when a job j_k overruns its deadline, the job is allowed to continue its execution, and no subsequent jobs are released until j_k completes its execution.

Assume that job j_k, released at time a_k, finishes its execution at time $f_k = a_k + 3.7\,p$. In this case, three subsequent jobs are skipped. During the three control periods in which j_k is still pending completion, no new job is released and the actuator outputs a control command that is in line with the actuation mode, either hold or zero. When j_k completes its execution, the controller state is updated and the control command is computed using the sensor value that was retrieved at time a_k, i.e., depending on whether the sensor message at time a_k was received or not. In this case, the new control signal is sent to the actuator at the end of the control period, at time instant $a_k + 4\,p$.

The closed-loop state vector for the continue and kill case can be reused to describe the system evolution of the continue and skip case, i.e., $\tilde{x}_k^{\mathsf{CS}} = [x_k^{\mathrm{T}}, z_k^{\mathrm{T}}, u_k^{\mathrm{T}}, \hat{y}_{k-1}^{\mathrm{T}}]^{\mathrm{T}}$. The closed-loop matrices are shown in Equation (8). Intuitively, there are 16 possible outcome configurations since $\Sigma(\mathcal{C}^{\mathsf{S}})$ contains four outcomes rather than two, i.e., $2^2 \cdot 4 = 16$. We adopt the same notation as for the kill case with $\Delta_{\mathcal{A}} = I$ or $\Delta_{\mathcal{A}} = 0$ to indicate, respectively, the hold and zero actuation mode.

When $c = \mathsf{R}$ or $c = \mathsf{N}$, the sensor message's outcome is irrelevant for the system dynamics. This follows since the control task does not poll the sensor channel for messages if it is still executing the body of the control algorithm (unless its outcome is M, since it is the first period that experiences an overrun). Again, note that many matrices are identical, i.e.,

$$\Lambda_{\mathsf{FMT}} = \Lambda_{\mathsf{FMF}}, \quad \Lambda_{\mathsf{TMT}} = \Lambda_{\mathsf{TMF}}$$
$$\Lambda_{\mathsf{TNT}} = \Lambda_{\mathsf{TNF}} = \Lambda_{\mathsf{FNT}} = \Lambda_{\mathsf{FNF}}$$
$$\Lambda_{\mathsf{FRT}} = \Lambda_{\mathsf{TRT}}, \quad \Lambda_{\mathsf{FRF}} = \Lambda_{\mathsf{TRF}},$$

which can be used to simplify the stability analysis below.

C. Markov Jump Linear Systems Analysis

While the evolution of the system dynamics is governed by the matrices presented Section III-B, the discrete state evolution of the closed-loop system is probabilistic and depends on the outcome of s, c, and a.

We can represent the discrete state evolution with a Markov chain, in which each state encodes one of the possible matrices that govern the discrete-time evolution of the system. The transition probabilities of the Markov chain depend on ρ_s, ρ_c, and ρ_a, respectively the probability of not receiving sensor data correctly, the probability of not completing the calculation of the control signal within one period, and the probability of not receiving the actuator data correctly. As an example, the probability that in one period we transition to the state in which no faults occurred, $s = \mathsf{T}$, $c = \mathsf{H}$, $a = \mathsf{T}$ is $(1 - \rho_s)\,(1 - \rho_c)\,(1 - \rho_a)$. In such a discrete state, the discrete-time dynamics evolve according to Λ_{THT}. For compactness, we use the notation 1_x to denote $(1 - \rho_x)$, e.g., $1_s = (1 - \rho_s)$.

For the kill case, in principle there are 8 states in the Markov chain (stemming from the 8 possible matrices), but the equivalence between two pairs of matrices reduces the discrete states in which the system can be found to 6, corresponding to the closed-loop matrices Λ_{THT}, Λ_{FHT}, Λ_{THF}, Λ_{FHF}, Λ_{FMX} and Λ_{TMX}, where X is used to indicate that the outcome is irrelevant in this specific case.

We can then define a Markov chain whose transition probabilities are indicated on the left side of Table I. The transition matrix is fully connected, as from each state it is possible to reach any other state. Also, given the nature of the kill action, every iteration of the control loop is independent, and therefore the probability to reach any state in the Markov chain is the same, regardless of the current state.

This is not true for the skip case. In fact, Rule 1 enforces that H can only occur after either another H or R. Also, N has to directly follow M, and a R can only follow a M or N. This implies that the transition matrix of the Markov chain for the continue and skip case is not fully connected and there are constraints on the reachable states. The transition matrix of

TABLE I

MARKOV CHAIN TRANSITION PROBABILITIES FOR CONTINUE AND KILL (LEFT) AND CONTINUE AND SKIP (RIGHT). HERE, $1_x = (1 - \rho_x)$.

	Λ_{THT}	Λ_{FHT}	Λ_{THF}	Λ_{FHF}	Λ_{FMX}	Λ_{TMX}
Λ_{THT}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$
Λ_{FHT}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$
Λ_{THF}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$
Λ_{FHF}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$
Λ_{FMX}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$
Λ_{TMX}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$

	Λ_{THT}	Λ_{FHT}	Λ_{THF}	Λ_{FHF}	Λ_{FMX}	Λ_{TMX}	Λ_{XNX}	Λ_{XRT}	Λ_{XRF}
Λ_{THT}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$	0	0	0
Λ_{FHT}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$	0	0	0
Λ_{THF}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$	0	0	0
Λ_{FHF}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$	0	0	0
Λ_{FMX}	0	0	0	0	0	0	ρ_c	1_c1_a	$1_c\rho_a$
Λ_{TMX}	0	0	0	0	0	0	ρ_c	1_c1_a	$1_c\rho_a$
Λ_{XNX}	0	0	0	0	0	0	ρ_c	1_c1_a	$1_c\rho_a$
Λ_{XRT}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$	0	0	0
Λ_{XRF}	$1_s1_c1_a$	$\rho_s1_c1_a$	$1_s1_c\rho_a$	$\rho_s1_c\rho_a$	$\rho_s\rho_c$	$1_s\rho_c$	0	0	0

the Markov chain for the continue and skip case is reported on the right side in Table I.

The transition probabilities are here treated as independent identically distributed (i.e., iid) random variables. Typically this is not the case in real systems. For example, if the sensor message is lost, the probability of the control task overrunning its deadline likely increases. It is possible to treat said cases using the theory of *conditional probabilities*, i.e., probabilities that depend on the outcome of another event. Due to space limitations, we don't enter into details on this matter here. However, we remark that the analysis method presented in this paper does not require probabilities to be iid, and can be generalised to also include the case when the probabilities are conditional, by changing the values of ρ_s, ρ_c, and ρ_a in a state-dependant manner, i.e., making the values different in a given state. This is in particular interesting for ρ_c, as a deadline miss that follows another miss for the skip case is less likely than the first deadline miss to occur.

The combination of the Markov system's state evolution and the closed-loop dynamics form a (autonomous) Markov Jump Linear System [10]. The convergence of Markov Jump Linear Systems can be analysed using different tools, and in particular there are two main notions of stability: *mean stability* and *mean square stability*. Mean stability corresponds to convergence in probability, while the second notion, mean square stability, corresponds to almost sure convergence.[4]

In both cases, we analyse the expected value of the state x_k, i.e., $\mathbb{E}[x_k]$, and determine whether it converges to a specific value or not. Mean stability analyses only whether the expected value of the system state converges or not. Mean square stability, on the other hand, analyses not only whether the expected value of the discrete-time system state converges, but also whether its covariance goes to zero; thus, implying *almost sure convergence*, i.e., the probability of the system state converging goes to 1 and the probability of the variance of the state going to zero goes to 1. The last part means that mean square stability also constrains the expected value of $\mathbb{E}[x_k x_k^{\mathrm{T}}]$. If the Markov Jump Linear System is mean square stable, it also implies that the system is mean stable. In our analysis we want to enforce the notion of mean square stability.

[4]Other notions of stability also exist, like stochastic stability and mean square exponential stability. However, if a system is mean square stable it is also mean stochastically stable as well as mean square exponentially stable. Almost sure convergence is hence a *stronger* notion of stability.

Testing for mean square stability implies calculating the eigenvalues of the operator Ψ, that defines the evolution of the Markov Jump Linear System's covariance matrix [10],

$$\Psi = (\Pi^{\mathrm{T}} \otimes I_{n^2})\,\mathrm{blkdiag}\,(\,\{\Lambda_{sca}^{\mathrm{T}} \otimes \Lambda_{sca}\}_{sca}). \tag{9}$$

Here, $\otimes$ represents the Kronecker product, Π is the Markov chain transition matrix specified in Table I, I_{n^2} is the identity matrix of size n^2, where $n = n_x + n_z + n_u + n_y$ is the order of the closed-loop matrices Λ_{sca}, and the last term is a block diagonal Kronecker product of the matrices in the possible closed-loop system realisations.

A Markov Jump Linear System is mean square stable if and only if

$$\sigma(\Psi) = \max|\,\mathrm{eig}\,(\Psi)\,| < 1. \tag{10}$$

Hence, we analyse the mean square stability of the system by constructing the operator Ψ and calculating its eigenvalues. Thus, the probabilistic stability of the closed-loop system under IO channel dropouts and computational overruns can be derived.

IV. EVALUATION

In this section we apply the Markov Jump Linear Systems stability analysis presented in Section III to three case studies lifted from the literature on controllers that experience faults:

- In Section IV-A we apply our analysis to an automotive cruise control system controlled with a state-feedback controller, taken from [15];
- In Section IV-B we address a Furuta pendulum controlled using a state-feedback controller, analysed in [31]; and
- In Section IV-C, we analyse a ball and beam process controlled by a Linear-Quadratic-Gaussian (LQG) controller, taken from [41].

We assume that both computational overruns and IO channel message losses are Bernoulli distributed [34], i.e., that the outcomes s, c, and a are independent and identically distributed random variables. We denote with ρ_s the probability of losing a message on the sensor channel; with ρ_c the probability of the control task overrunning a deadline; and with ρ_a the probability of losing a message on the actuator channel.

For each case study, we perform the following three sets of experiments:

(i) We fix *two* of the probabilities ρ_s, ρ_c, and ρ_a to 0 and vary the remaining one between 0 and 1 (excluded) with a step of 0.01, i.e., $\rho_x \in \{0,\, 0.01,\, \ldots\, 0.99\}$, $x \in \{s, c, a\}$.

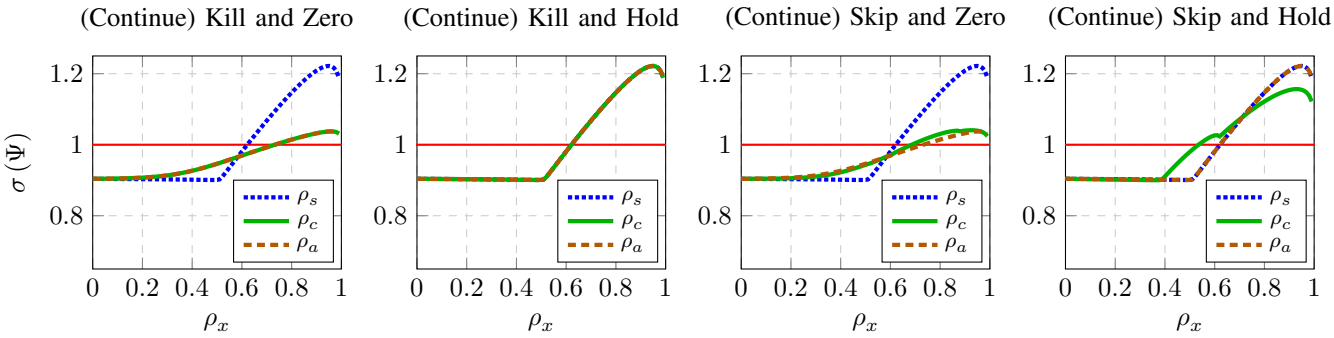

Fig. 2. Results of Experiment (i) on the cruise control plant [15] for different deadline handling strategies and actuation modes.

(ii) We fix *one* of the probabilities ρ_s, ρ_c, and ρ_a to 0 and vary the remaining two between 0 and 1 (excluded) with a step of 0.01.

(iii) We analyse the closed-loop system dynamics when the sensor channel experiences 15% traffic loss, the controller executes in a busy real-time operating system and thus overruns 40% of its deadlines, and the actuator channel has a dropout of 5% of its packets. This corresponds to setting $\rho_s = 0.15$, $\rho_c = 0.4$, and $\rho_a = 0.05$.

We analyse controllers that are implemented with kill and skip as deadline overrun handling strategy and zero and hold as actuation modes. For all experiments, we calculate $\sigma(\Psi)$ to determine the closed-loop mean square stability according to Equation (10), where Ψ is the Markov Jump Linear System's covariance matrix calculated according to Equation (9).

A. Automotive Cruise Control Evaluation

Ghosh et al. [15] present a method to derive a fault-tolerant state-feedback controller to address stochastic computational faults. Additionally, the performance and stability of said controller are validated on the model of an automotive cruise control system, which we denote with $\mathcal{P}_1$. The plant is inherently stable, i.e., $\sigma(A) < 1$. In this paper, we analyse the automotive cruise control system controlled by a baseline state-feedback controller $\mathcal{C}_1$, also presented in [15].

$$\mathcal{P}_1 : \begin{cases} x_{k+1} & = A\,x_k + B\,u_k \\ y_k & = x_k \end{cases}$$

where

$$A = \begin{bmatrix} 1 & 0.01 & 0 \\ -0.0003 & 0.9997 & 0.01 \\ -0.0604 & -0.0531 & 0.9974 \end{bmatrix}, \; B = \begin{bmatrix} 0.0001 \\ 0.0001 \\ 0.0247 \end{bmatrix}$$

and

$$\mathcal{C}_1 : u_{k+1} = \begin{bmatrix} -872.54 & -131.49 & -10.097 \end{bmatrix} x_k.$$

Experiment (i): Figure 2 shows the results of experiment (i). Each plot corresponds to a particular strategy combination (e.g., kill and zero) and the x-axis shows the probability that is varied, $\rho_x \in \{0, 0.01, \ldots, 0.99\}$ (while the other probabilities are set to 0), while the y-axis shows the result of the analysis, $\sigma(\Psi)$. A value of $\sigma(\Psi)$ that exceeds 1 implies that the system with the given ρ_x is not mean square stable.

When the sensor channel's message loss probability ρ_s increases, so does the magnitude of the closed-loop system's eigenvalues, no matter the choice of strategy to handle the deadline miss and the actuation mode. In the case of the sensor dropout plot, $\rho_s \neq 0$ and $\rho_c = \rho_a = 0$. Hence, the controller will never miss a deadline and the actuator will always output a new control signal, implying that dropouts on the sensor channel are invariant to the choice of deadline overrun strategy and actuation mode.

When $\rho_s = \rho_c = 0$ and $\rho_a \neq 0$, the stability of the system depends on the choice of actuation mode, but is agnostic to the deadline overrun strategy. When the hold actuation mode is favoured, the actuator outcome is comparable to missing a sensor packet and continuing the controller execution, since the sensor messages always arrive ($\rho_s = 0$) and the controller never overruns its deadline ($\rho_c = 0$). Additionally, we deduce that the automotive cruise control system tolerates a higher probability of losing actuator messages before going unstable under the zero actuation mode than under hold. This likely follows from the system being inherently stable.

For the kill overrun strategy, the eigenvalue of Ψ with the largest magnitude is identical for computational overruns and lost actuator messages. Since the controller is a static state-feedback law, as soon as a deadline is hit a new control signal is computed without any residual problems originating from a diverged control state (state-feedback controllers are stateless). However, when computational overruns occur and the scheduler has adopted the skip strategy, the computed control signal is based on old plant states, and thus negatively impacts the robustness of the control system.

The shape of the curve representing the computational overruns, i.e., the case when $\rho_c \neq 0$, $\rho_s = \rho_a = 0$, is strongly dependent on the system dynamics. When the probability of overrunning a deadline goes to 1, the system runs in *open-loop*, i.e., without any feedback. Since the plant is stable, running in open-loop system would preserve the stability of the system, with $\sigma(\Psi)$ being close to 1. In fact, the *switching* between meeting and overrunning deadlines is the cause of destabilisation in the closed-loop system. This is consistent

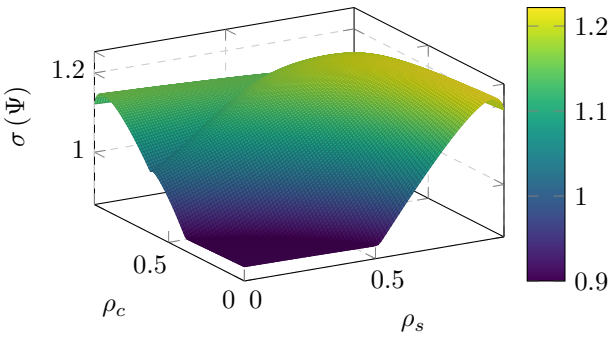

Fig. 3. Results of Experiment (ii) on the cruise control plant [15] for (continue) skip and hold as deadline handling strategy and actuation mode.

TABLE II
RESULTS OF EXPERIMENT (III) ON THE AUTOMOTIVE CRUISE CONTROL.

	Kill+Zero	Kill+Hold	Skip+Zero	Skip+Hold
$\sigma(\Psi)$	0.9313	0.9006	0.9274	0.9638

with observations presented in [40] about how the closed-loop robustness can improve with an increased number of computational overruns. We do however emphasise that despite the value of $\sigma(\Psi)$ decreasing, it is still above 1, thus indicating that the system is *not* mean square stable.

Experiment (ii): Figure 3 displays a 3d plot of $\sigma(\Psi)$ when both sensor packet dropouts and computational overruns may occur, but the actuator messages are always delivered correctly, i.e., $\rho_s \neq 0$, $\rho_c \neq 0$, and $\rho_a = 0$, for skip and hold.[5]

The x- and y-axes show the probabilities ρ_s and ρ_c, while the z-axes corresponds to $\sigma(\Psi)$. It is possible to recognise the curves of $\rho_s = 0$ and $\rho_c = 0$ from Figure 2. For configurations where both sensor channel losses and computational overruns are present, the system robustness is degraded. As an example, individually when $\rho_s = 0.51$ or $\rho_c = 0.51$ the system is stable (see Figure 2), but when both values are set to 0.51, the system is unstable, as can be seen in Figure 3.

Experiment (iii): Table II shows the results of Experiment (iii). Regardless of deadline overrun strategy and actuation mode, the Markov Jump Linear System is stable when $\rho_s = 0.15$, $\rho_c = 0.4$, and $\rho_a = 0.05$. The results confirm that the cruise control system is robust to simultaneous occurrences of multiple fault types. It is interesting to note that the outcome configuration $(\rho_s, \rho_c, \rho_a) = (0, 0.4, 0)$ leads to $\sigma(\Psi) = 0.9108$ for the skip and hold strategy. In other words, despite the faults on the IO channels appearing to be inconsequential for the system stability (for $\rho_s < 0.5$ and $\rho_a < 0.5$) in Figure 2, they significantly affect the dynamics of the system. In fact, perturbing the probabilities on the IO channels from Experiment (iii) by 8% to $(\rho_s, \rho_c, \rho_a) = (0.23, 0.4, 0.13)$, the system is unstable with a value of $\sigma(\Psi) = 1.0014$.

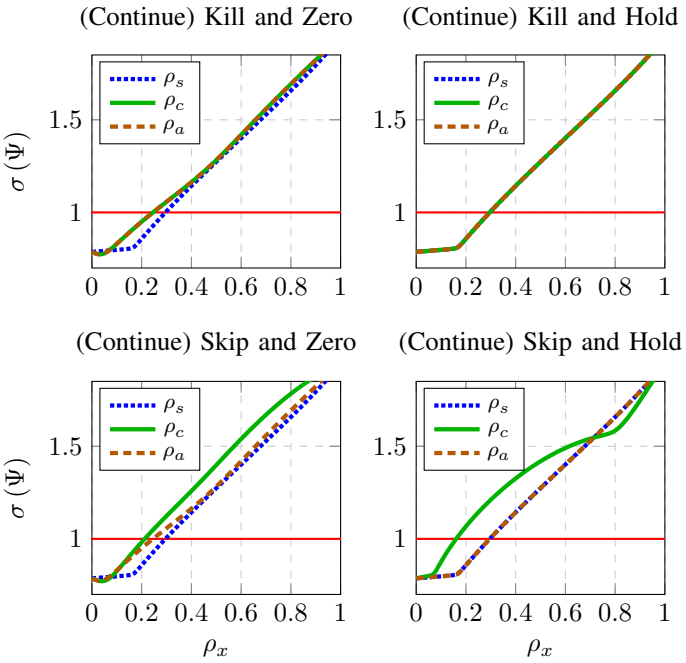

Fig. 4. Results of Experiment (i) with the Furuta pendulum [31] and different deadline handling and actuation mode strategies.

B. Furuta Pendulum Evaluation

Pazzaglia et al. [31], present a systematic method for analysing control system performance when the control task is subject to deadline overruns. The paper describes the Furuta pendulum $\mathcal{P}_2$ as a case study, and controls it using a state-feedback controller. We modify the controller $\mathcal{C}_2$ presented in [31] to account for the one-step delay introduced by the LET paradigm. An important distinction from the cruise control system is that the Furuta pendulum is inherently unstable, i.e., $\sigma(A) > 1$.

$$\mathcal{P}_2 : \begin{cases} x_{k+1} &= A\,x_k + B\,u_k \\ y_k &= x_k \end{cases}$$

where

$$A = \begin{bmatrix} 1 & 0.0036 & 0.0188 & -0.0007 \\ 0 & 1.2282 & -0.0332 & 0.0503 \\ 0 & 0.0266 & 0.0081 & -0.0032 \\ 0 & 3.7230 & -0.2448 & 0.2794 \end{bmatrix}, \; B = \begin{bmatrix} 0.1047 \\ 0.0429 \\ 1.2801 \\ 0.3159 \end{bmatrix}$$

and

$$\mathcal{C}_2 : u_{k+1} = \begin{bmatrix} -0.400 & 47.341 & -1.399 & 2.033 \end{bmatrix} x_k.$$

Experiment (i): The results of Experiment (i) are presented in Figure 4. Similarly to the cruise control system from the previous section, the packet losses on the sensor channel are agnostic to the deadline overrun strategy and actuation mode. Additionally, the effects on the system's stability from losses on the actuation channel only differ from the sensor channel

[5]Experiments with different configurations or strategies do not provide additional insights and are hence not reported due to space limitations.

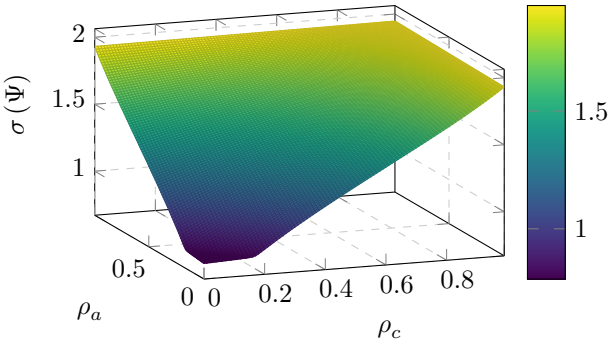

Fig. 5. Results of Experiment (ii) on the Furuta pendulum [31] for (continue) kill and hold as deadline handling strategy and actuation mode.

TABLE III
RESULTS OF EXPERIMENT (III) ON THE FURUTA PENDULUM EXAMPLE.

	Kill+Zero	Kill+Hold	Skip+Zero	Skip+Hold
$\sigma(\Psi)$	1.2159	1.2210	1.3254	1.3942

messages in the zero case. The main difference between the two examples come from how the computational overruns affect the system stability. Recall that plant $\mathcal{P}_1$ is inherently stable, while $\mathcal{P}_2$ is not. This extends also to the robustness of the closed-loop systems. In particular, the controller for the Furuta pendulum $\mathcal{C}_2$ is less robust to deadline overruns than its counterpart $\mathcal{C}_1$ for the cruise control system, i.e., the largest ρ_c that keeps $\sigma(\Psi)$ below 1 is smaller for the Furuta pendulum than for the cruise control.

Experiment (ii): Figure 5 shows $\sigma(\Psi)$ with respect to variations of ρ_c and ρ_a for the Furuta pendulum when kill and hold are adopted as deadline handling strategy and actuation mode. It is apparent that there are very few stable configurations. In fact, from the 10 000 tested configurations, only 512 satisfy the stability constraint $\sigma(\Psi) < 1$. Even with small values, like $\rho_c = 0.17$ and $\rho_a = 0.16$, the system is not mean square stable.

Experiment (iii): Table III shows the results of Experiment (iii). Unsurprisingly, the system is unstable for the outcome configuration $(\rho_s, \rho_c, \rho_a) = (0.15, 0.4, 0.05)$.

C. Ball and Beam Evaluation

Vreman et al. [41] propose a controller implementation method that aims at improving the performance of systems where the controller is subject to probabilistic deadline overruns. The implementation method is evaluated on a physical ball and beam plant, $\mathcal{P}_3$, controlled using a linear-quadratic-Gaussian (LQG) controller $\mathcal{C}_3$:

$$\mathcal{P}_3 : \begin{cases} x_{k+1} &= A\,x_k + B\,u_k \\ y_k &= C\,x_k + D\,u_k \end{cases}, \mathcal{C}_3 : \begin{cases} z_{k+1} &= F\,z_k + G\,y_k \\ u_{k+1} &= H\,z_k + K\,y_k \end{cases}$$

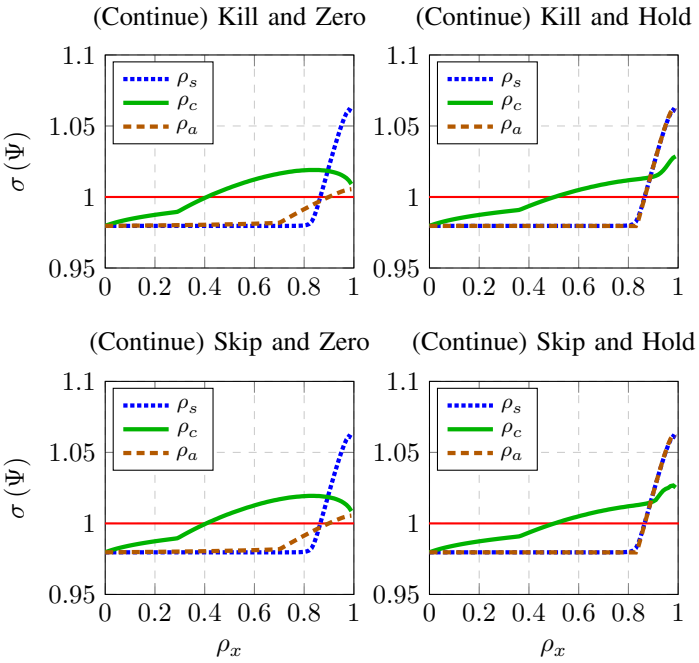

Fig. 6. Results of Experiment (i) with the ball and beam [41] and different deadline handling and actuation mode strategies.

where

$$A = \begin{bmatrix} 1 & 0 & 0 \\ -0.1 & 1 & 0 \\ -0.0005 & 0.01 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0.045 \\ -0.0023 \\ -7.5 \cdot 10^{-6} \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D = 0,$$

$$F = \begin{bmatrix} 0.709 & 0.054 & 0.041 \\ 0.011 & 0.997 & -0.219 \\ 0.004 & 0.010 & 0.934 \end{bmatrix}, \quad G = \begin{bmatrix} 0.152 & 0.001 \\ -0.104 & 0.217 \\ -0.004 & 0.066 \end{bmatrix}$$

$$H = \begin{bmatrix} -2.433 & 1.201 & 0.562 \end{bmatrix}, \quad K = \begin{bmatrix} -0.672 & 0.368 \end{bmatrix}.$$

Note that the ball and beam plant is modelled as a triple integrator, i.e., there are three eigenvalues with magnitude $\sigma(A) = 1$, making the system unstable.

Experiment (i): Figure 6 shows the results of Experiment (i). Unlike the Furuta pendulum and cruise control system, there now exists a clear distinction between the three different cases of Experiment (i). It is evident that the sensor, actuator, and controller all affect the Markov Jump Linear System stability individually, and can thus not be seen as equivalent outcomes.

The contrast between the behaviour of the ball and beam system and the prior two setups come from the controller dynamics in $\mathcal{C}_3$. Both $\mathcal{C}_1$ and $\mathcal{C}_2$ are stateless controllers. On the other hand, the LQG controller $\mathcal{C}_3$ does have an internal state, meaning that if the controller overruns a deadline or an IO channel message is lost, the effect of such an event will impact the system negatively for some time after the event. Despite the introduced controller dynamics, the closed-loop ball and beam system $\mathcal{G}_3$ appear to be robust to both IO channel message losses (up to ρ_s and ρ_a around 0.85) and

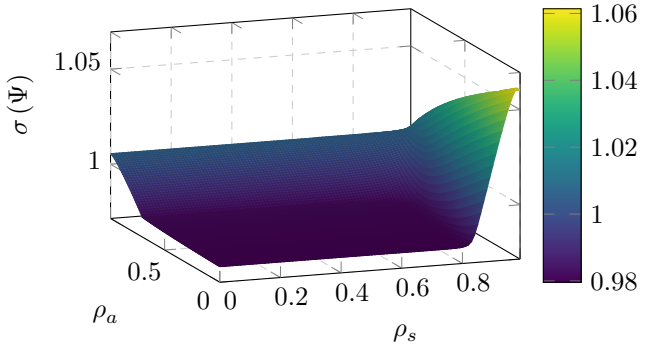

Fig. 7. Results of Experiment (ii) on the ball and beam [41] for (continue) kill and zero as deadline handling strategy and actuation mode.

TABLE IV
RESULTS OF EXPERIMENT (III) ON THE BALL AND BEAM EXAMPLE.

	Kill+Zero	Kill+Hold	Skip+Zero	Skip+Hold
$\sigma(\Psi)$	1.0000	0.9936	1.0002	0.9936

computational overruns (up to ρ_c around 0.35 for the zero actuation mode and 0.5 for hold actuation mode).

Experiment (ii): The surface plot in Figure 7 shows $\sigma(\Psi)$ with varying ρ_s and ρ_a, assuming that the controller always succeeds in meeting its computational deadline ($\rho_c = 0$) and that the zero actuation mode is adopted. It is easy to see that even for high values of both ρ_s and ρ_a, the system remains mean square stable as long as the controller does not miss its deadlines. This is likely a result of the controller being designed with robustness as a design objective.

Experiment (iii): Table IV presents the results acquired from Experiment (iii) on the ball and beam system. The results indicate that the choice of actuation mode is affecting $\sigma(\Psi)$ more than the choice of deadline overrun strategy. In fact, both implementations employing the zero actuation mode are unstable, while if the hold actuation mode is selected, the closed-loop system is mean square stable under the tested probabilities. When we increase the probability of the IO channels experiencing packet drops, the zero actuation mode quickly becomes unstable, while the hold actuation mode can tolerate up to 75% packet drops on *both* the sensor and actuator channels simultaneously before becoming mean square unstable, i.e., before $\sigma(\Psi) \geq 1$. This likely follows from the model of the system being a triple integrator, where zeroing the output signal in case of a packet loss or computational overrun drives the system state away from the desired value. Instead, holding the previous control command keeps the integrator state close to its ideal value.

V. Related Work

In recent years, probabilistic analysis techniques for real-time systems are becoming more prominent [11]. In particular, a few interesting analysis methods have been developed to compute the probability that specific tasks miss their deadlines, e.g., [38]. In [8], the authors propose a method to safely estimate the deadline miss rate of tasks in a uniprocessor system under a preemptive fixed-priority scheduling policy. For mixed-criticality systems, [28] introduce a probabilistic analysis method for analysing worst-case execution time distributions, and in turn worst-case deadline miss probabilities, under fixed-priority preemptive scheduling. An efficient and accurate convolution-based approach to calculating deadline miss probabilities is introduced in [39]. The authors of [27] further contribute to the area by proposing a method for down-sampling the random variables in order to improve space and time complexity of the analysis methods.

The ratios provided by the deadline miss probability analysis can be utilised to improve the design of control systems. In [34], the authors derive an optimal controller (control law and estimator) for a networked control system where packet arrivals are modelled stochastically. The authors of [9] propose a stabilising controller for networked control systems subject to stochastic network delays and packet dropouts. In [30], a robust controller is derived for the case when the control task can experience deadline misses stochastically.

Generally, fault-tolerant controllers are designed to counteract one specific type of fault, e.g., sensor dropouts *or* deadline overruns. To analyse whether a system is robust to the specific type of fault or not, different analysis methods have been proposed. A stability analysis method for control systems subject to weakly-hard packet dropouts is proposed in [18]. In [26], the authors propose a method for analysing the stability of real-time control systems where the control task is subject to consecutive deadline overruns.

VI. Conclusion and Future Work

There exists a common misconception that analysing a control system's robustness to packet losses on the network is equivalent to having the control algorithm overrun its timing budget, and vice versa. This paper proposes an approach to analyse real-time control systems and their stability properties when subject to multiple types of faults. In particular, we analyse *both* dropouts on the IO communication channels and computational overruns of the task executing the control algorithm, making the analysis more comprehensive than the state-of-the-art alternatives.

We envision that the analysis method and the corresponding experimental campaign will be used to improve future analysis methods and correct any misconceptions about how faults interact in computer-controlled systems. Finally, the paper brings the control analysis closer to the state-of-practice compared to the research literature, because it relies on a probabilistic failure model. In industrial setups, it is in fact easier to get estimates of the probability of certain events from testing campaigns, rather than to extract complex (but deterministic) guarantees like the validity of a weakly-hard constraint.

References

[1] L. Ahrendts, S. Quinton, T. Boroske, and R. Ernst. Verifying weakly-hard real-time properties of traffic streams in switched networks. In *30th Euromicro Conference on Real-Time Systems, ECRTS 2018, July*

3-6, 2018, Barcelona, Spain, volume 106 of *LIPIcs*, pages 15:1–15:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[2] G. Bernat, A. Burns, and A. Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50:308–321, 2001.

[3] W. P. Blair Jr. and D. D. Sworder. Feedback control of a class of linear discrete systems with jump parameters and quadratic cost criteria. *International Journal of Control*, 21(5):833–841, 1975.

[4] P. Bolzern, P. Colaneri, and G. De Nicolao. Markov jump linear systems with switching transition rates: Mean square stability with dwell-time. *Automatica*, 46(6):1081–1088, 2010.

[5] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency*. Plenum Publishing Co., 2005.

[6] A. Cervin. Analysis of overrun strategies in periodic control tasks. In *16th IFAC World Congress*. Elsevier, 2005.

[7] K.-H. Chen, N. Ueter, G. v. der Brüggen, and J.-J. Chen. Efficient computation of deadline-miss probability and potential pitfalls. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 896–901, 2019.

[8] K.-H. Chen, G. Von Der Brüggen, and J.-J. 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, 2018.

[9] M. Cloosterman, L. Hetel, N. van de Wouw, W. Heemels, J. Daafouz, and H. Nijmeijer. Controller synthesis for networked control systems. *Automatica*, 46(10):1584–1594, 2010.

[10] O. Costa, M. Fragoso, and R. Marques. *Discrete-Time Markov Jump Linear Systems*. Applied probability. Springer, 2005.

[11] R. I. Davis and L. Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *Leibniz Transactions on Embedded Systems*, 6(1), 2019.

[12] M. C. F. Donkers, W. P. M. H. Heemels, N. van de Wouw, and L. Hetel. Stability analysis of networked control systems using a switched linear systems approach. *IEEE Transactions on Automatic Control*, 56(9):2101–2115, 2011.

[13] Y. Fang and K. Loparo. Stochastic stability of jump linear systems. *IEEE Transactions on Automatic Control*, 47(7):1204–1208, 2002.

[14] S. K. Ghosh, S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty. Design and validation of fault-tolerant embedded controllers. In J. Madsen and A. K. Coskun, editors, *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 1283–1288. IEEE, 2018.

[15] S. K. Ghosh, S. Dey, D. Goswami, D. Mueller-Gritschneder, and S. Chakraborty. Design and validation of fault-tolerant embedded controllers. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1283–1288, 2018.

[16] D. Goswami, D. Mueller-Gritschneder, T. Basten, U. Schlichtmann, and S. Chakraborty. Fault-tolerant embedded control systems for unreliable hardware. In *International Symposium on Integrated Circuits*, 2014.

[17] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. Giotto: A time-triggered language for embedded programming. *Proceedings of the IEEE*, 91(1), 2003.

[18] M. Hertneck, S. Linsenmayer, and F. Allgöwer. Stability analysis for nonlinear weakly hard real-time control systems. *IFAC-PapersOnLine*, 53(2), 2020.

[19] M. Kauer, D. Soudbakhsh, D. Goswami, S. Chakraborty, and A. Annaswamy. Fault-tolerant control synthesis and verification of distributed embedded systems. In *Design, Automation & Test in Europe Conference Exhibition*, 2014.

[20] D. Liberzon. Finite data-rate feedback stabilization of switched and hybrid linear systems. *Automatica*, 50(2):409–420, 2014.

[21] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 1319–1324 vol.2, 2002.

[22] Q. Ling and M. Lemmon. Robust performance of soft real-time networked control systems with data dropouts. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 2, 2002.

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

[24] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):4661, 1973.

[25] Y. Ma, Y. Wang, S. Cairano, T. Koike-Akino, J. Guo, P. Orlik, and C. Lu. A smart actuation architecture for wireless networked control systems. 2018.

[26] M. Maggio, A. Hamann, E. Mayer-John, and D. Ziegenbein. Control-system stability under consecutive deadline misses constraints. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, volume 165 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[27] F. Marković, A. V. Papadopoulos, and T. Nolte. On the Convolution Efficiency for Probabilistic Analysis of Real-Time Systems. In *33rd Euromicro Conference on Real-Time Systems*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[28] D. Maxim, R. I. Davis, L. Cucu-Grosjean, and A. Easwaran. Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017.

[29] M. Ohlin. *Feedback Linux Scheduling and a Simulation Tool for Wireless Control*. PhD thesis, Department of Automatic Control, 2006.

[30] P. Pazzaglia, C. Mandrioli, M. Maggio, and A. Cervin. DMAC: Deadline-Miss-Aware Control. In *31st Euromicro Conference on Real-Time Systems*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

[31] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. D. Natale. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In S. Altmeyer, editor, *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[32] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems (3rd Ed.)*. Prentice-Hall, Inc., USA, 1997.

[33] L. Schenato. To zero or to hold control inputs with lossy links? *IEEE Transactions on Automatic Control*, 54(5):1093–1099, 2009.

[34] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry. Foundations of control and estimation over lossy networks. *Proceedings of the IEEE*, 95(1):163–187, 2007.

[35] M. Schinkel, W.-H. Chen, and A. Rantzer. Optimal control for systems with varying sampling rate. In *Proceedings of the 2002 American Control Conference*, volume 4, pages 2979–2984 vol.4, 2002.

[36] J. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo. Implications of classical scheduling results for real-time systems. *Computer*, 28(6):16–25, June 1995.

[37] Y. Sun and M. D. Natale. Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks. *ACM Trans. Embed. Comput. Syst.*, 16(5s):171:1–171:19, 2017.

[38] G. von der Brüggen, N. Piatkowski, K. Chen, J. Chen, K. Morik, and B. B. Brandenburg. Efficiently approximating the worst-case deadline failure probability under EDF. In *42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021*, pages 214–226. IEEE, 2021.

[39] G. von der Brüggen, N. Piatkowski, K.-H. Chen, J.-J. Chen, and K. Morik. Efficiently Approximating the Probability of Deadline Misses in Real-Time Systems. In *30th Euromicro Conference on Real-Time Systems*, Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[40] N. Vreman, A. Cervin, and M. Maggio. Stability and performance analysis of control systems subject to bursts of deadline misses. In *33rd Euromicro Conference on Real-Time Systems (ECRTS)*, volume 196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[41] N. Vreman, C. Mandrioli, and A. Cervin. Deadline-miss-adaptive controller implementation for real-time control systems. In *IEEE 28th Real-Time and Embedded Technology and Applications Symposium*, pages 13–26, 2022.

[42] W. Wang, P. Mishra, and A. Gordon-Ross. Dynamic cache reconfiguration for soft real-time systems. *ACM Transactions on Embedded Computer Systems*, 11(2), July 2012.