# Checking Scheduling-induced Violations of Control Safety Properties

Anand Yeolekar[1], Ravindra Metta[1], Clara Hobbs[2], and Samarjit Chakraborty[2]

[1] TCS Research, India & TUM Germany {anand.yeolekar,r.metta}@tcs.com
[2] The University of North Carolina at Chapel Hill, USA
{cghobbs,samarjit}@cs.unc.edu

**Abstract.** Cyber-physical systems (CPS) are typically implemented as a set of real-time control tasks with periodic activation. When a control task misses it's deadline, policies for handling deadline miss – *e.g.* delayed scheduling of the task instance – may still lead the CPS into an unsafe or sub-optimal state. We present a technique for *exact* checking of such control safety and reachability properties, for a class of CPS, under common deadline miss handling and control update policies. In particular, we propose a joint encoding of control and scheduling behaviour as a satisfiability-modulo-theory formulation and a novel abstraction-refinement procedure with incremental solving to scale the analysis. Case studies with realistic systems show the utility of our approach.

**Keywords:** control · scheduling · verification · abstraction · refinement

## 1 Introduction

CPS controllers are typically designed as a set of real-time tasks assuming ideal conditions, such as all tasks meet their deadlines, for ease of design, by abstracting away the implementation details. However, when the tasks are finally implemented in software, the control performance might deviate from the expected behaviour due to factors such as control task missing deadlines due to transient overload on the processor. When a control task instance misses its deadline, then depending on how the CPS is configured to handle deadline misses, the corresponding control computation may be *skipped* or *delayed* causing a potential deviation from the expected behaviour. Such intermittent deviation from expected behaviour, depending on when it occurs and by what amount, may in turn lead to control safety or reachability violation.

For example, consider the F1Tenth car model [14], where the controller is designed to steer the car along a predetermined path, without hitting an obstacle. Only some deadline miss patterns, coupled with selected choice of initial state of the car, will result into a collision (see Fig. 5). In this work, we focus on *analysing the interaction* between control and scheduling leading to such violations.

Existing analysis techniques assume a simplified scheduling model, such as bounding the maximum number of *consecutive* deadline misses [18, 10], or restrict the scheduling behaviour by not admitting non-determinism in task specification, or non-preemption of tasks. However, a precise bound accounting for

all feasible scheduling behaviours may not be easily identifiable for a given task set implementing the CPS. As a result, these techniques tend to be *pessimistic*, meaning the assumed worst-case deadline miss pattern, based on the bound, might never occur for a real system. Further, bugs that occur based on the interaction between control and scheduling layers are often subtle, near-impossible to reproduce by analyzing separately control or scheduling.

Therefore, to establish system correctness with respect to control properties, an analysis that **precisely maps** task runs containing all permitted sequences of deadline misses to control behaviour is needed, especially if the control or task specification admits non-determinism. Such an analysis helps CPS designers (i) to gauge the impact of scheduling parameters on control performance, and (ii) gain insight into the interplay of control evolution, scheduling policy, and strategies for handling deadline misses. Towards this end, we propose an approach to check scheduling-induced violations of control safety and reachability properties.

**Summary of our approach:** Given a (discrete) control system, a set of tasks realizing the controller, and an analysis horizon $h$ indicating length of control evolution, we construct an *abstract* model of the system behaviour (control evolution and runs of the tasks), which admits all feasible system behaviour as well as some infeasible (spurious) behaviour. We check this model for violation of specified control properties using a constraint solver, which reports a *witness* on finding a violation. If the witness is spurious (an infeasible task run or control trajectory), we iteratively *refine* the model to *block* the spurious witnesses, until either a genuine witness is obtained or no more witnesses exist, proving that the control property holds on the original system. Our main contributions are:

**Encoding:** we propose a Satisfiability-Modulo-Theory (SMT) encoding that *abstracts* control and scheduling by *relaxing* certain constraints on their behaviour, which is then composed together with the property to be checked. Our encoding admits non-determinism at the control layer (arbitrary initial states), and scheduling layer such as delayed release (jitter) and variable execution times.

**Refinement:** we construct *blocking implications* from spurious witnesses to refine the abstraction, and utilize the *incremental analysis* feature of SMT solvers to efficiently analyze reasonably sized controllers and task sets.

**Tool:** we implemented the above abstraction and the refinement scheme to check control property violations, supporting four common combinations of deadline miss handling and control update strategies, and one static and one dynamic non-preemptive scheduling policy.

**Related work:** Encoding of control and schedule to assess and correct impact of scheduling anomalies on control performance has been studied [17] when a set of periodic tasks with implicit deadlines is not schedulable, and to systematically adjust the task periods to achieve schedulablity. [5] combines the control and timing models as hybrid automata and verifies with Space Ex model checker [6].

The impact of timing uncertainty, such as deadline misses, on control has been studied [18, 10, 12, 20], but broadly focused on analysis of control stability *i.e.* whether control trajectories converge to an equilibrium point. However, as

Table 1: Symbolic variables used in the SMT encoding

| Notation | Type | Description |
|---|---|---|
| $x_{j,k}$ | Real | $j$-th dimension of plant state at $k$-th time step |
| $u_k$ | Real | Control update value at $k$-th step |
| $r_k^i, s_k^i, e_k^i, d_k^i$ | Int | release, start, end, and deadline times of the $k$-th job of $i$-th task |

observed in [1], a stable control system might still violate safety properties, hence the need for an approach to check safety properties. [15] proposes a rich state-based representation for capturing deadline misses and measure their impact on control performance. [16] presents a static scheduling strategy that guarantees control performance while smartly saving resources. Another approach of automatically adapting the control system to deadline misses to guarantee performance is proposed in [19], along with worst-case stability analysis. However, these approaches need a bound to be specified on the number of consecutive deadline misses possible.

The effect of control trajectories going outside safe regions in CPS has also been studied and remedial actions were proposed [21, 4] for fixed priority scheduling and controller co-design. [3] dynamically extends the period of control tasks, based on historical measurements, to reduce power consumption and accommodate increased resource demands from other components.

In most works, a model of deadline misses needs to be provided by user, which may sometimes help scale the analysis better than our approach. However the main limitations of these approaches are: (i) *extracting* the assumed model of deadline misses from the task specification, and (ii) unavoidable pessimism due to worst case assumptions. In contrast, our approach faithfully models task runs and control evolution, for *precise* analysis.

## 2 System model and encoding

### 2.1 Control system model and evolution

A discrete control system describing the plant model is defined as:

$$x_{k+1} = Ax_k + Bu_k \qquad u_k = R - Kx_k \tag{1}$$

where $x \in \mathbb{R}^{n\times 1}$ is the discrete state vector, $n \geq 1$ is the control system dimension; $k \in \mathbb{N}$ denotes the discrete steps of evolution; $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times 1}$ are matrices specifying the discrete-time plant model with timestep $\delta$. The control input $u \in \mathbb{R}$ is computed using a state feedback vector $K \in \mathbb{R}^{1\times n}$, and an optional reference value $R \in \mathbb{R}$. The initial state $x_0$ must lie in a user-specified interval $[\underline{X}_0, \overline{X}_0]$ along each dimension.

Our encoding approach unrolls the closed-loop discrete control system described in Eqn. 1 up to the user-specified bound $h$. To unroll, we introduce

symbolic variables $x_{j,k}$ corresponding to the control states, and $u_k$ corresponding to the control action update, where $j$ ranges over the dimension of the control system $1, 2, \ldots, n$, and $k$ ranges over the discrete steps of evolution $0, 1, \ldots, h$. We first construct the constraints on the initial plant state as:

$$\phi_{init} \coloneqq \forall j : \; \underline{X}_{j,0} \leq x_{j,0} \leq \overline{X}_{j,0} \tag{2}$$

Then, symbolically encoding the trajectories that could originate from any point in the initial set, we construct constraints on the control state variables as:

$$\phi_{traj} \coloneqq \bigwedge_{k=0}^{h} \left( \bigwedge_{i=1}^{n} \; x_{i,k+1} = \sum_{j'=1}^{j} A_{i,j'} x_{j',k} + B_i u_k \right) \tag{3}$$

We defer the explanation of the timing model and associated control update modeling $u_k$ to Sec. 2.4. Given the analysis horizon $h$, the control safety and reachability properties over the trajectories are:

$$\begin{aligned} \phi_{prop} &\coloneqq \underline{X}_{j,h} \leq x_{j,h} \leq \overline{X}_{j,h} && \text{(reachability)} \\ \phi_{prop} &\coloneqq \underline{X}_{j,k} \leq x_{j,k} \leq \overline{X}_{j,k} \;, \; 0 \leq k \leq h && \text{(safety)} \end{aligned} \tag{4}$$

where $[\underline{X}_{j,h}, \overline{X}_{j,h}]$, $[\underline{X}_{j,k}, \overline{X}_{j,k}]$ denote the user-specified reach and safety intervals (or safety pipes), respectively, for $j$-th dimension.

### 2.2 Task specification

The controller is realized in software via a set of tasks $\mathcal{T}$ that includes the control and auxiliary tasks *e.g.* loggers, communication, etc. We currently support non-preemptive earliest-deadline-first (NP-EDF) representing dynamic priority scheduling, and rate-monotonic (NP-RM), representing static priority scheduling[3], under unicore setting. A task $\tau_i \in \mathcal{T}$ is defined as $\big(O, J, \underline{E}, \overline{E}, P\big)$, where $i$ is a unique task id, $O$ is the task offset, $J$ denotes release jitter faced by task instances, $\underline{E}$ and $\overline{E}$ denote the best- and worst-case execution times of the tasks respectively, and $P$ denotes the period. We assume $\tau_0$ corresponds to the controller task with period set to the discretization timestep: $P^0 = \delta$.

We refer to task instances as *jobs*. Release time of the $k$-th job spawned by $\tau_i$ is denoted by $r^i_k$. Due to release jitter, the instant of job release lies in the interval $[kP^i + O^i, \; kP^i + O^i + J^i]$. We denote the start, end and deadline time of the job as $s^i_k, e^i_k$ and $d^i_k$. We assume task deadlines are implicit, thus $d^i_k = O^i + (k+1)P^i$, and a deadline *miss* occurs when $e^i_k > d^i_k$. Under CONTINUE policy, jobs are eventually scheduled even if they miss deadline, and under KILL, jobs are aborted in case of *conservative* deadline miss (*i.e.*, a job is aborted if its execution does not begin by $d^i_k - \overline{E^i}$). Under NP-RM, jobs of the same task are scheduled in the order of release. Finally, jobs are released, scheduled and terminate at *discrete* time points.

[3] While our method can be adapted to handle preemptions, we focus on NP scheduling for ease of presentation and leave the extension as future work.

**Definition 1.** *A run of the task set is a timed sequence of jobs, $\langle \ldots, (i, k, s^i_k, e^i_k), \ldots \rangle$, respecting the given scheduling policy and deadline miss strategy.*

We assume the scheduling is *work-conserving i.e.* a ready job must be scheduled as soon as the processor is available. Observe that multiple runs of the task set are possible due to (i) release jitter experienced by each job, (ii) variable execution budget leading to non-deterministic termination time for each job, and (iii) arbitrary selection of equal-priority ready jobs. These runs can have varying impact on the control performance and need to be analyzed rigorously.

### 2.3 An abstraction for task runs

We explain how to encode the set of runs of the task set. Our approach spawns jobs of all tasks up to the time instant $h \times \delta$ and we encode runs of the task set as a logical formula. There is no explicit modeling of the scheduler; the operational semantics of the scheduling process, *e.g.*, the scheduler's run queue, tasks moving from sleep to ready state, etc. are modeled implicitly in the formula.

From the task specification (Sec. 2.2) we construct constraints on the symbolic variables (Table 1) associated with each spawned job as:

$$\begin{aligned} \phi_{runs} \coloneqq \forall\ (i,k):\ & r^i_k \le s^i_k \ \wedge\ kP^i + O^i \le r^i_k \le kP^i + O^i + J^i \wedge e^i_k \le s^i_{k+1} \\ & \wedge\ \underline{E}^i \le e^i_k - s^i_k \le \overline{E}^i \text{ (under CONTINUE)} \\ \wedge\ (s^i_k + \overline{E}^i \le d^i_k \Rightarrow \underline{E}^i \le e^i_k - s^i_k \le \overline{E}^i) & \wedge (s^i_k + \overline{E}^i > d^i_k \Rightarrow e^i_k = s^i_k) \text{ (under KILL)} \end{aligned} \quad (5)$$

These constraints restrict the release, start and end times of jobs as per the task specification and deadline miss policy, however, they exclude the scheduling policy and work conservation at this stage of modeling. While this helps to keep the constraints concise and tractable, it introduces an *abstraction* with respect to the set of valid runs of the task set (admits all valid runs as well as spurious ones) as defined in Defn. 1. In Sec. 3, we will restore precision by using refinements to prune away the spurious behaviour *i.e* invalid task runs.

### 2.4 Control action update modeling

We admit ZERO and HOLD policies for control update $u$, where ZERO signifies applying $u = 0$ when the corresponding control task instance misses deadline, and HOLD signifies applying the previous value. Fig. 1 illustrates the simplified logical execution timing (LET) model assumed in this work and the associated control action updates, under HOLD semantics. Plant sensing and actuation happens *instantaneously* at fixed discrete time points $k\delta$, irrespective of the scheduling of tasks[4]. As shown in the figure, job $k-1$, spawned at time $(k-1)\delta$, reads the plant state $x_{k-1}$ at the beginning of its execution, processes the data, and writes

[4] We assume a time-triggered hardware implementation of sensing/actuation, outside the scheduling purview, with values stored in buffers accessed by the control task.

an actuation value at termination. This actuation value is applied to the plant at the *next* time step $k\delta$, and corresponds to the control action update $u_k$. We assume $u_0 = 0$ (open loop for first step). On a deadline miss, *e.g* instance $k$ missing deadline, $u_{k+1}$ is matched to $u_k$ (due to HOLD). Observe that instance $k$, which missed its deadline, is scheduled in the *next* slot $[(k+1)\delta, (k+2)\delta)$, enabling it to read the relatively *fresher* plant state $x_{k+1}$. Instance $k+1$ is scheduled in its own slot $[(k+1)\delta, (k+2)\delta)$ but misses its deadline, instance $k+2$ meets its deadline, and both write sequentially to the actuation buffer in the same time slot, corresponding to control action update $u_{k+3}$. In such a case, we assume the actuation buffer is *overwritten* by the fresher value.

Updates are thus *delayed* when the controller is realized in software. We model this by constructing *conditional control update* constraints as:

$$\begin{aligned}
\phi_u \coloneqq \forall k : u_k = 0,\ \text{if } k = 0 \\
\wedge\ e^0_{k-1} \le d^0_{k-1} \Rightarrow u_k = R - \sum_{j=1}^{n} K_j x_{j,k-1} \text{ (under C}\textsc{ontinue}) \\
\wedge\ s^0_{k-1} + \overline{E}^0 \le d^0_{k-1} \Rightarrow u_k = R - \sum_{j=1}^{n} K_j x_{j,k-1} \text{ (under K}\textsc{ill}) \\
\wedge\ s^0_{k-1} + \overline{E}^0 > d^0_{k-1} \Rightarrow u_k = u_{k-1} \text{ (under H}\textsc{old}\text{-K}\textsc{ill}) \\
\wedge\ s^0_{k-1} + \overline{E}^0 > d^0_{k-1} \Rightarrow u_k = 0 \text{ (under Z}\textsc{ero}\text{-K}\textsc{ill})
\end{aligned} \quad (6)$$

Notice that, under CONTINUE policy, the above constraints enforce control update computation when deadlines are *met*, but leaves the control update *unconstrained* on a deadline miss. This introduces an *abstraction* with respect to control updates. This is necessary at this stage of modeling, as we do not know *statically* how many jobs could miss being *consecutively scheduled* all together in any run of the given task set *i.e.* how much to "look back" from the current step to pick the *preceding* control task instance execution, to use that value as the freshest, when encoding the control update. Additionally, this helps in keeping the control action constraints tractable and concise. In Sec. 3.4, we will restore precision by refining control updates. Note that under KILL policy, control updates are always precisely computed (there is no abstraction).

**Definition 2 (Trajectory).** *A (discrete) trajectory of the control system is a sequence of values of state variables $\langle \ldots, (k, x_1, \ldots, x_j), \ldots \rangle$, originating from a valid initial state, ordered on the evolution step counter $k$, respecting the state and control equations 1 and 6.*

### 2.5 Composing control and scheduling models

From the encodings for the control trajectories from Eqns. 2 and 3, task runs from Eqn. 5, control update from Eqn. 6, and the control property from Eqn. 4, we construct the system composition as:

$$\phi_{sys} \coloneqq \phi_{init} \wedge \phi_{traj} \wedge \phi_u \wedge \phi_{runs} \wedge \neg\phi_{prop} \quad (7)$$

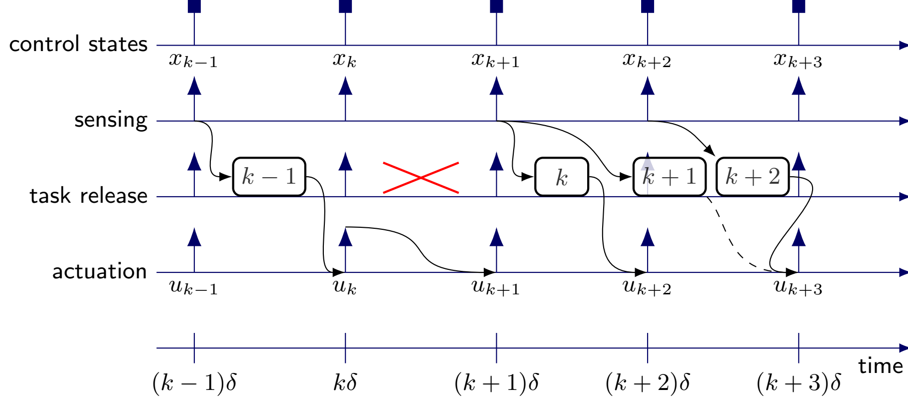



Fig. 1: Timing model illustrating sensing, task release and actuation, under HOLD

## 3 Refining the abstraction

Consider a solution to $\phi_{sys}$ reported by an SMT solver, that assigns concrete values to all the symbolic variables in the formula. The solution is parsed to extract (i) the run of the tasks consisting of a sequence of jobs, termed $\sigma_{run}$, and (ii) the control trajectory, sorted on the step counter $k$, termed $\sigma_{traj}$. If $\sigma_{run}$ satisfies Defn. 1, we have a run generated from the abstract $\phi_{sys}$ that precisely maps to a *concrete run* of $\mathcal{T}$. Similarly, if $\sigma_{traj}$ satisfies Defn. 2, we have a control trajectory, generated from the abstract $\phi_{sys}$, that precisely maps to a *concrete trajectory* of the control system.

However, if either $\sigma_{run}$ or $\sigma_{traj}$ violate their respective definitions, we have a *spurious* trace leading to property violation. To block such a trace from $\phi_{sys}$, we identify the causes of non-compliance within the definitions. For Defn. 1 the causes can be overlapping jobs, scheduling policy violation, or work conservation violation, and for Defn. 2, unconstrained control update due to deadline miss.

### 3.1 Overlapping jobs

Suppose job $(i, j)$ overlaps with $(i', j')$ in $\sigma_{run}$, with $s^i_j \leq s^{i'}_{j'}$. This is possible as the abstraction does not prevent overlaps *upfront*. Observe that though in this trace $(i, j)$ preceded $(i', j')$, there can be a run of $\mathcal{T}$ with the precedence reversed. Thus, to block this overlap as witnessed in this trace, we construct:

$$B_{ov} \coloneqq (s^i_j \leq s^{i'}_{j'} \;\wedge\; s^{i'}_{j'} < e^i_j) \Rightarrow e^i_j \leq s^{i'}_{j'} \tag{8}$$

This implication when conjuncted with $\phi_{sys}$ blocks *this* particular pair of jobs from overlapping again in *any* trace, under the premise that $(i, j)$ precedes $(i', j')$.

### 3.2 Schedule violation

Suppose job $(i, j)$ precedes $(i', j')$ in $\sigma_{run}$, but this precedence violates the scheduling policy. Under NP-EDF, $(i, j)$ can precede $(i', j')$ if and only if the

deadline of $(i, j)$ is *no later than* that of $(i', j')$, or $(i, j)$ is scheduled strictly before $(i', j')$ is released. Thus we construct the blocking implication[5]:

$$B_{sv} \coloneqq s^i_j < s^{i'}_{j'} \Rightarrow (d^i_j \leq d^{i'}_{j'} \ \vee \ s^i_j < r^{i'}_{j'}) \qquad \text{(under NP-EDF)} \tag{9}$$

Conjuncting $B_{sv}$ with $\phi_{sys}$ blocks the scheduling violation caused by *this* pair of jobs in *any* trace, under the premise that $(i, j)$ precedes $(i', j')$.

### 3.3 Work conservation violation

Here, the processor cannot idle in the presence of a ready job. We assume that $\sigma_{run}$ is free of overlapping jobs, easily achieved by repeatedly refining $\phi_{sys}$ with $B_{ov}$ implications. There are two cases to analyze: (a) processor idling immediately after release of job $(i, j)$, implying that $s^i_j = r^i_j$, and (b) idling post termination of some job $(i', j')$ within the waiting time of $(i, j)$, implying $s^i_j = e^{i'}_{j'}$.

Observe, however, there can be runs of $\mathcal{T}$ with different jobs preceding $(i, j)$, which raises the question: what is the set of jobs preceding $(i, j)$ across all runs? This set, denoted $prec^i_j$, is *conservatively* estimated as follows: Intuitively, jobs released earlier *and* having higher priority will *always* precede $(i, j)$ in all runs, and vice versa. Importantly, this set of jobs can be identified *statically* based on their period and deadline. Then, the complement of this set, characterized by a lack of static precedence guarantee, forms $prec^i_j$. Formally, consider job $(i', j'), i' \neq i$. If $d^{i'}_{j'} < d^i_j \ \wedge \ j'P^{i'} + O^{i'} + J^{i'} <= jP^i + O^i$ (and vice-versa) *does not hold* (under NP-EDF)[6], then $(i', j') \in prec^i_j$. Observe that *prec* sets need to be computed only once per job violating work conservation.

The concrete starting instant of the processor idle interval, which is $r^i_j$ in case (a), serves to *partition* the set of jobs $prec^i_j$, as witnessed in $\sigma_{run}$, into: (i) a *prefix* subset of jobs scheduled prior to $r^i_j$, and (ii) a *suffix* subset of jobs scheduled post $r^i_j$. Since there are no job overlaps in $\sigma_{run}$, we are guaranteed that *prefix* and *suffix* are mutually exclusive. Thus, we construct the blocking implication for case (a) to preventing processor idling as:

$$B_{wc_a} \coloneqq (\ r^i_j < s^i_j \wedge \bigwedge_{(i',j') \in prefix} e^{i'}_{j'} < r^i_j \wedge \bigwedge_{(i',j') \in suffix} s^{i'}_{j'} > r^i_j\ ) \Rightarrow s^i_j = r^i_j \tag{10}$$

Here, the antecedent captures the context witnessed in $\sigma_{run}$ that: (i) job $(i, j)$ had a non-zero waiting time, (ii) some jobs that could precede $(i, j)$ were scheduled prior to $r^i_j$, (iii) the remaining jobs that could precede $(i, j)$ were scheduled post $r^i_j$. Under these premises, the consequent enforces work conservation.

Similarly for case (b), $e^{i'}_{j'}$ partitions *prec*, changing the consequent to $s^i_j = e^{i'}_{j'}$.

---

[5] Under NP-RM, priority (period) must be higher (lower): $P_i \leq P_{i'} \ \vee \ s^i_j < r^{i'}_{j'}$

[6] Under NP-RM, this is $P^{i'} < P^i$

### 3.4 Unconstrained control updates

The basic idea for refining unconstrained control updates is to locate the latest control job that was scheduled in $\sigma_{run}$, compute the control update issued by this job (if not already done), and use this as the freshest value. Observe that we cannot always pick the control update issued by the *preceding* job: From Fig. 1, job $k+1$ missed its deadline, leading to an unconstrained $u_{k+2}$. However, here, we cannot pick $u_{k+1}$ to match $u_{k+2}$, as job $k$ did get scheduled and successfully terminated before the instant $(k+2)\delta$, thereby issuing a *fresher* control update that must be matched with $u_{k+2}$. We discuss the various cases below.

**Case 1:** Suppose job $(0,n)$ missed its deadline $d_n^0$ in $\sigma_{run}$, enabling a spurious assignment to $u_{n+1}$ in $\sigma_{traj}$. Then, suppose examining $\sigma_{run}$ leads us to a job $(0,m), m < n$, as the closest control job that was scheduled (and thus terminated) prior to the instant $d_n^0$. Then, $u_{n+1}$ should have matched $u_{m+1}$, based on HOLD. Now, if job $(0,m)$ has met its deadline in $\sigma_{run}$, then $u_{m+1}$ is already computed (and the concrete value is reflected in $\sigma_{traj}$). This allows us to build the blocking implication for this first case as:

$$B_{uu_a} \coloneqq (\ e_n^0 > d_n^0 \wedge e_m^0 \leq d_m^0 \wedge e_{m+1}^0 > d_n^0\ ) \Rightarrow u_{n+1} = u_{m+1} \tag{11}$$

Here, the antecedent captures the context that (i) job $(0,n)$ missed deadline, (ii) job $(0,m)$ is the closest preceding job to the time instant $d_n^0$ (through $e_{m+1}^0 > d_n^0$) and met deadline. This case is illustrated in Fig. 1 with job $k$ missing deadline and job $k-1$ meeting its deadline, enforcing $u_{k+1}$ to match $u_k$.

**Case 2:** Consider that the preceding job $(0,m)$ too missed its deadline in $\sigma_{run}$, and hence $u_{m+1}$ is not computed, as defined in Eqn. 6. We have to locate the "scheduling slot" in which $(0,m)$ started execution, encode computation of the the corresponding control update, and match with $u_{n+1}$. Recall that control task instances read the control state at the beginning of their execution. Suppose, by examining $\sigma_{run}$, we observe that $m'P^0 \leq s_m^0 < (m'+1)P^0$, with $m \leq m' \leq n$. In other words, job $(0,m)$ was scheduled in a slot (interval of length $P^0$) that begins at time $m'P^0$. Then, job $(0,m)$ must have read the control state available in this slot, which allows us to construct a blocking implication that computes the correct control action as:

$$\begin{aligned} B_{uu_b} \coloneqq (\ & e_n^0 > d_n^0 \wedge e_m^0 > d_m^0 \wedge e_m^0 \leq d_n^0 \wedge e_{m+1}^0 > d_n^0 \\ & \wedge m'P^0 \leq s_m^0 < (m'+1)P^0\ ) \Rightarrow u_{n+1} = R - \sum_{j'=1}^{j} K_{j'} x_{j',m'} \end{aligned} \tag{12}$$

Here, the antecedent captures the conditions that: (i) jobs $(0,n)$ and $(0,m)$ missed their deadlines, (ii) job $(0,m)$ is closest one to be scheduled prior to the time instant $d_n^0$, and (iii) job $(0,m)$ was scheduled in the slot beginning at time $m'P^0$. The consequent constrains the control update to pick the control state $x_{j,m'}$ *i.e.* state at time $m'P^0$. This case is illustrated in Fig. 1, with jobs $k+1$ and $k$ missing their respective deadlines. Job $k$, however, is scheduled in the next slot post release ($m' = k+1$), and terminates before the time instant

$(k+2)\delta$, leading to $u_{k+2}$ matching the control update issued by job $k$, albeit reading the control state $x_{k+1}$ instead of $x_k$.

**Case 3:** The last special case that no preceding job is found (all preceding jobs missed deadline) can be handled similarly by enforcing $u_{n+1} = u_0$.

### 3.5 Correctness of refinement

**Theorem 1.** *Based on Eqns. 8–12, each refinement step removes only spurious runs and/or trajectories from the set of solutions of* $\phi_{sys}$.

*Proof (sketch).* $B_{ov}$ (Eqn. 8) ensures that pairs of jobs do not overlap and does not obstruct any run of $\mathcal{T}$. $B_{sv}$ (Eqn. 9) prevents incorrect scheduling of pairs of jobs by blocking such spurious runs.

$B_{wc}$ prevents processor idling in the presence of ready jobs, by "moving" the waiting job appropriately, thus blocking the spurious run. Note that the *prec* set (Sec. 3.3), by construction, *soundly over-approximates* the set of jobs that could precede the violating job *in all runs* of $\mathcal{T}$. Thus, the antecedent in $B_{wc}$ (Eqn. 10) is *guaranteed* to cover all possible spurious cases involving this job, *i.e.*, the refinement is *complete* with respect to work-conservation violation.

$B_{uu}$ prevents unconstrained control updates by processing the trace $\sigma_{run}$ to locate the closest preceding job and computes the control update issued by this job (if not already computed in $\sigma_{traj}$). The cases presented in Eqns. 11 and 12 guarantee that the *freshest* update is identified within $\sigma_{traj}$ and matched to restore precision. Thus *in all cases*, the refinement implications block spurious behaviour or traces of $\phi_{sys}$ that *do not* constitute runs of $\mathcal{T}$ or $\mathcal{C}$. These scenarios are the only causes of spuriousness in $\phi_{sys}$.

## 4 Tool design

Fig. 2 depicts our tool implementation of the abstraction (Sec. 2) and refinement (Sec. 3) using Python 3.8 and Z3 4.8.12. The tool accepts (i) control specification $(A, B, K, X_0)$, analysis horizon $h$, (ii) safety and reachability sets of plant states, (iii) task specification, and scheduling policy. Jobs are spawned up to the analysis horizon and symbolic variables (Table 1) are introduced for each job. Formulas $\phi_{traj}$ and $\phi_{runs}$ are constructed and conjuncted along with the control property of interest. If Z3 reports unsatisfiability, the property holds.

If Z3 reports a witness, we parse it to extract assignments to the symbolic variables and reconstruct the task run and control trajectory. Internally, Z3 tracks the set of formulas on symbolic variables using a *context* stack. Refinement iterations *incrementally* add blocking clauses to the current context, leveraging the incremental analysis capability of Z3. The refinement loop is split into two phases, catering to the two sources of abstraction, $\phi_{traj}$ and $\phi_{runs}$. During experiments, we observed that a majority of the refinements were required for pruning spurious task runs, as compared to spurious control trajectories. Hence, we specifically built a separate loop for quickly refining $\phi_{runs}$ with the advantage

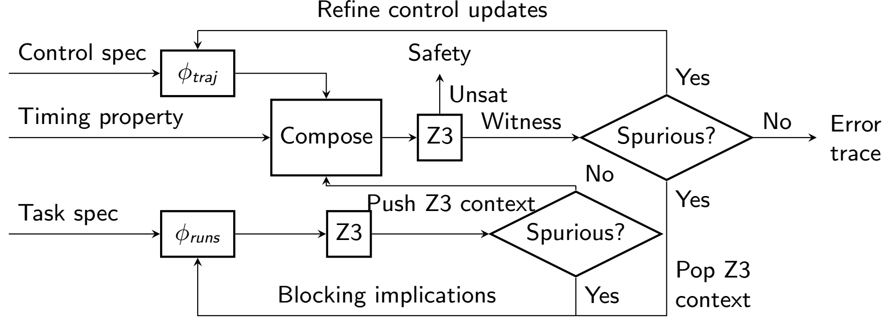



Fig. 2: Tool design

that the composition of $\phi_{traj}$ and $\phi_{runs}$, which yields the larger formula $\phi_{sys}$ and consequently larger state space to be explored, is built and analyzed only after a valid task run is obtained, thus boosting Z3 performance.

Refinement loops interface using context pushing and popping API from Z3. Just before the composition step, the context is pushed *i.e.* saved on Z3 stack. Constraints from $\phi_{traj}$ are then added to the context, encoding the entire system $\phi_{sys}$. Spurious control updates are processed according to Sec. 3.4. While refining $\phi_{sys}$, if we obtain a spurious task run, it is likely that several iterations of refinements over $\phi_{runs}$ will be needed (as evidenced during experiments). At this point, the presence of constraints from $\phi_{traj}$ in the solver context is an unnecessary burden for the solver. Consequently, the saved context is popped out, flushing out $\phi_{traj}$ constraints and restoring $\phi_{runs}$ refined upto the last good point, thereby boosting the solver performance.

## 5 Case study 1: DC motor control model

Consider a DC motor speed model adapted from [11], specified by the model:

$$x_{k+1} = \begin{bmatrix} 0.9058 & 0.09617 & 0 \\ 0.01923 & 1.021 & 0 \\ 0 & 0 & 0 \end{bmatrix} x_k + \begin{bmatrix} -0.009742 \\ -0.2021 \\ 1 \end{bmatrix} u_k$$

$$u_k = \begin{bmatrix} -0.219719 & -0.942677 & 0.184469 \end{bmatrix} x_k$$

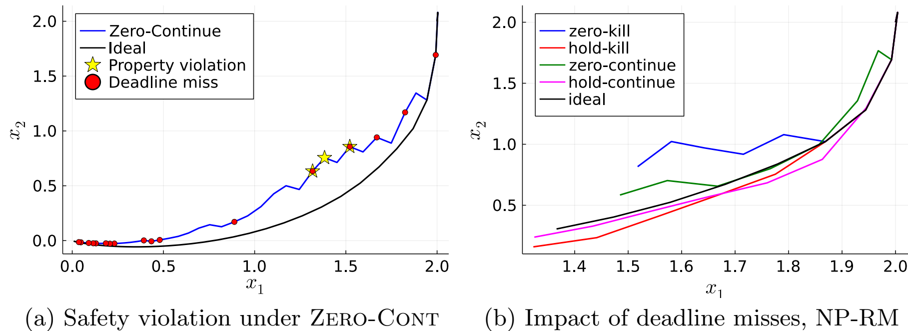
This discrete-time model has a period of 100 ms. We assume a synthetic task set, consisting of 5 tasks, implementing the controller, described in Table 2. Task $\tau_0$ is the controller task, thus $P^0 = 100$ ms. We considered the following properties:
**Property 1:** Safe angular velocity of motor: $x_1^{ideal} - 0.3 \leq x_1 \leq x_1^{ideal} + 0.3$ *i.e.* the angular velocity must not deviate by more than 0.3 units from the corresponding ideal (*i.e* no deadline miss) control states, at each step of evolution.
**Property 2:** Safe current through armature: $x_2 \leq x_2^{ideal} + 0.5$ *i.e.* the current through armature must not rise by more than 0.5 units from the corresponding ideal current values, at each step of evolution.

Table 2: Synthetic task set for DC Motor controller

| id | offset | period | [BCET, WCET] | jitter |
|---|---|---|---|---|
| $\tau_0$ | 0 | 100 | $[15, 30]$ | 2 |
| $\tau_1$ | 0 | 100 | $[15, 30]$ | 2 |
| $\tau_2$ | 0 | 100 | $[15, 30]$ | 2 |
| $\tau_3$ | 10 | 400 | $[15, 30]$ | 2 |
| $\tau_4$ | 10 | 500 | $[20, 40]$ | 2 |

Table 3: Task set for RC Network controller

| id | offset | period | [BCET, WCET] | jitter |
|---|---|---|---|---|
| $\tau_0$ | 0 | 100 | [6,13] | 2 |
| $\tau_1$ | 0 | 50 | [6,16] | 2 |
| $\tau_2$ | 0 | 100 | [15,30] | 2 |
| $\tau_3$ | 2 | 250 | [8,16] | 2 |
| $\tau_4$ | 6 | 100 | [15,30] | 2 |
| $\tau_5$ | 2 | 500 | [5,15] | 0 |




(a) Safety violation under ZERO-CONT (b) Impact of deadline misses, NP-RM

Fig. 3: Analysis of DC Motor control system.

We applied the tool to check these safety properties over the DC motor system. Analysis over 50 steps, under ZERO-CONTINUE strategy and NP-EDF schedule, revealed violation of the properties. Fig. 3a shows the trajectory reported by the tool, with Property 1 (angular velocity staying within specified bounds) violated at steps 15 and 16. Property 2 (current through armature within specified bounds) was reported violated at steps 9, 11 and 12. The task set run corresponding to this trajectory had a total of 17 deadline misses of the control task ($\tau_0$) spread over the 50 steps of evolution. The computation time was approximately 1.5 minutes, requiring 65 refinements. All experiments were executed on a laptop with Intel i5 processor, 16GB RAM and Ubuntu 20 OS.

**Comparing tool precision.** To illustrate improvement in precision in computing reach states using our tool (CONCH), we compared with a tool that computes a sound over-approximation of the reachable set (REACH) [8]. Unlike our method, this tool only explicitly models the control system, and over-approximates the possible scheduling behaviors with a constraint on the maximum number of consecutive deadline misses. This provides a baseline against which CONCH tool can be compared, illustrating the benefit of modeling the scheduling explicitly. For better computational efficiency, the REACH tool also over-approximates the reachable sets themselves, creating further pessimism that our method avoids.

Table 4: Reach upper bounds, under ZERO-KILL

| var | steps | ideal | REACH | CONCH |
|---|---|---|---|---|
| $x_1$ | 10 | 1.2724 | 1.9051 | 1.61 |
| $x_2$ | 10 | 0.2163 | 1.449 | 0.74 |
| $x_1$ | 20 | 0.4681 | 1.5075 | 0.81 |
| $x_2$ | 20 | -0.0628 | 1.0539 | 0.28 |

Table 5: Tool performance for 60 steps, NP-EDF schedule

| | $B_{ov}$ | $B_{sc}$ | $B_{wc}$ | $B_{uu}$ | R | Miss | Time |
|---|---|---|---|---|---|---|---|
| ZERO-KILL | 819 | 501 | 2531 | NA | 189 | 6 | 466 |
| HOLD-KILL | 819 | 501 | 2531 | NA | 189 | 6 | 657 |
| HOLD-CONT | 565 | 212 | 202 | 20 | 56 | 18 | 274 |
| ZERO-CONT | 565 | 212 | 202 | 20 | 56 | 18 | 379 |

We performed this comparison under ZERO-KILL strategy and NP-EDF schedule, with initial control states set to the point $x_0 = (2, 2, 0)$. Observe that under KILL and NP-EDF, this task set admits at most 2 consecutive deadline misses for $\tau_0$, and so this constraint is applied for the REACH tool. CONCH discovered the bounds by incrementing the ideal reach values in small steps and checking if the revised bound is violated, until it hit a safe value. Table 4 shows the **safe** reach upper bounds, for variables $x_1, x_2$, for 10 and 20 steps of evolution. Column "ideal" reports the value of states reached by the ideal trajectory (no deadline misses.) The safety bounds computed by REACH, with at most 2 consecutive deadline misses, is significantly over-approximated due to assuming a worst-case scheduling scenario of $k$-misses *every* $k + 1$ instances, which may not occur in practice, as illustrated by this task set.

**Illustrating deadline miss policies.** Fig. 3b illustrates the impact of the various deadline miss handling policies on control evolution, for this task set, under NP-RM. For KILL policy, the jobs that missed deadline were 0,5,6, and for CONTINUE policy, jobs 1,2 and 6 missed deadline. The graph zooms on the first 10 steps of control evolution, to illustrate the sets of control states (or alternately, segments of control evolution) that are more sensitive to deadline misses. Control behaviour under different strategies is impacted differently by similar sequences of deadline misses. We believe this analysis can help the control designer in uncovering finer insights into the interplay of scheduling policy, task parameters and strategies for deadline miss / control action update. Further, observe that for both NP-RM (Fig. 3b) and NP-EDF (Fig. 3a) policies, the maximum deviation from ideal behaviour generally occurs during the early steps of system evolution. This highlights the need to rigorously analyze small, transient segments of control evolution.

**Tool performance and insights.** Table 5 shows the scalability of the tool for 60 steps and the five tasks, for a custom reachability property, under NP-EDF schedule. The $B$ columns list the number of blocking implications mined across all iterations, column "R" lists refinements *i.e.* calls to the SMT solver Z3 [2], column "Miss" lists deadline misses witnessed in the property violation trace produced by the Z3. As seen from the table, tool performance is sensitive to the task set and deadline miss policy; this task set was crafted to admit a

large number of runs arising from non-deterministic scheduling choices, jitter and execution budget, in an attempt to showcase the tool's capability.

## 6 Case study 2: RC network control model

Consider an RC network model, adapted from [7], specified as:

$$x_{k+1} = \begin{bmatrix} 0.5495 & 0.0724 & 0.1616 \\ 0.01448 & 0.9332 & 0.02665 \\ 0 & 0 & 0 \end{bmatrix} x_k + \begin{bmatrix} 0.2166 \\ 0.02569 \\ 1 \end{bmatrix} u_k$$
$$u_k = \begin{bmatrix} 0.0977 & 0.2504 & 0.0781 \end{bmatrix} x_k$$

This discrete-time model has a period of 100 ms. We assume that the controller is implemented by a task set inspired from the real-life PapaBench [9] task set for an unmanned aerial vehicle, adapted for our setting. The adapted task set used for our experiment is described in Table 3.

For the RC network control system, we consider the safety property that maximum voltage across both capacitors does not exceed the ideal voltage by 0.1 units: $x_1 \leq x_1^{ideal} + 0.1 \wedge x_2 \leq x_2^{ideal} + 0.1$. The scheduling policy is set to NP-EDF and the strategy chosen is ZERO-CONTINUE.

Application of our tool for this system reveals property violation, shown in Fig. 4. The control jobs that missed deadlines are 2, 6, 8, 10, 11, 12, 14, 15. Variable $x_1$ violates the safety property at steps 5 and 10 within the 20-step analysis horizon. Notice that continuous deadline misses (*e.g.* jobs 10, 11, 12) cause more deviation from the ideal behaviour than isolated incidents of deadline miss. Depending on the control application, the deviation might be unacceptable, and thus this requires a precise analysis of scheduling and its impact on control.

The REACH tool, under the above setting, reported an *upper bound* on the deviation experienced by $x_1$ as 0.2715, whereas our tool CONCH reported a *tighter* bound of 0.15, which took 745 refinements and 150 seconds, and this safe bound was arrived at by incrementing and checking in steps of 0.01 units.

For HOLD-CONTINUE strategy, the tool reported that $x_1$ did not violate the property *i.e.* the maximum voltage for capacitor 1 stays within the given safe bounds over the analysis horizon. Proving safety took 579 refinements and approximately 2.5 minutes.

For KILL strategy, no control job misses deadline (other task instances miss deadline and are killed, allowing the control task to be always successfully scheduled within the analysis horizon). This again demonstrates that the task specification in combination with the strategies for handling deadline miss and control action updates can have significantly differing impact on control behaviour.

## 7 Case study 3: F1Tenth car model

Our final model captures the motion of an F1Tenth [14] model car, adapted for our setting (linearized, $x_1$ dimension dropped, discretized at 20ms), with

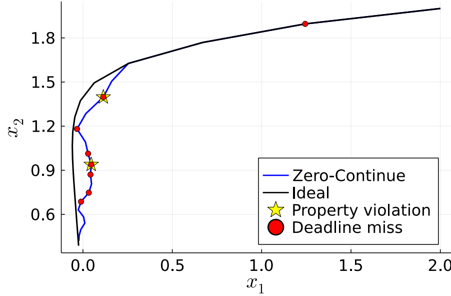



Fig. 4: Safety violating trajectory for RC Network, ZERO-CONT strategy

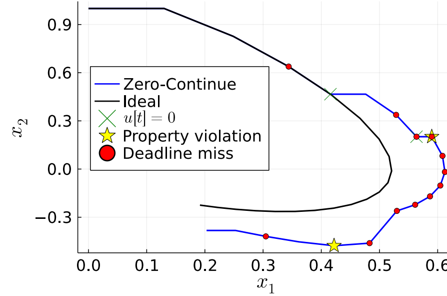



Fig. 5: Safety violation for F1Tenth car model

controller adapted from [13], as:

$$x_{k+1} = \begin{bmatrix} 1 & 0.13 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0.02559 \\ 0.3937 \end{bmatrix} u_k$$
$$u_k = \begin{bmatrix} 0.2935 & 0.4403 \end{bmatrix} x_{k-1}$$

The task set is adapted from the synthetic example presented in Tab. 2, where we drop task $\tau_4$, periods of tasks $\tau_0 - \tau_2$ are set to 20 and their execution times are set to [4,6], period of $\tau_3$ is set to 40 and execution time is set to [5,10]. The safety property of interest is that the steering angle should not deviate by more than 0.2 units from the ideal behaviour: $-0.2 \leq x_2 - x_{ideal} \leq 0.2$.

Under ZERO-CONTINUE strategy, CONCH reported property violation, as shown in Fig. 5. The task set run had a total of 12 deadline misses for the control task $\tau_0$, but ZERO control update occurred only twice in this run (since under CONTINUE, these jobs were eventually scheduled). Observe that the control trajectory violated *both* the upper and lower safety threshold, at steps 8 and 16, respectively. Interestingly, under KILL strategy, the bound was not violated. Proving property safety took 193 refinements and nearly 53 seconds.

## 8 Conclusions and Future Work

Our approach for *exact* checking of control properties, by jointly encoding control evolution and task scheduling under common deadline miss handling policies, could successfully check both safety and reachability properties that might be impacted due to scheduling issues of controller tasks, within practically acceptable time limits. Additionally, our tool can provide useful insights to CPS designers to: (i) Precisely compute control behaviour at step-wise granularity of evolution, (ii) Explore the impact of design choices *e.g.* ZERO-CONT vs. HOLD-KILL, and (iii) Explore the impact of task parameters on control *e.g.* release jitter. We believe this can address a large variety of practical problems involving control and scheduling interaction, which may be otherwise hard to reproduce

or debug. For future work, we plan to extend our encoding to model and analyze distributed CPS with three components: control, scheduling and network.

**Acknowledgement:** Hobbs and Chakraborty were funded by NSF grant 2038960.

## References


1. Abate, A., Bessa, I., Cattaruzza, D., Cordeiro, L., David, C., Kesseli, P., Kroening, D., Polgreen, E.: Automated formal synthesis of digital controllers for state-space physical plants. In: CAV (2017)
2. Bjørner, N., Phan, A., Fleckenstein, L.: $\nu$z - an optimizing smt solver. In: TACAS. pp. 194–199 (2015)
3. Dai, X., Burns, A.: Period adaptation of real-time control tasks with fp scheduling in cyber-physical systems. Journal of Sys. Arch. **103** (2020)
4. Dai, X., Zhao, S., Jiang, Y., Jiao, X., Hu, X.S., Chang, W.: Fixed-priority scheduling and controller co-design for time-sensitive networks. In: CAV (2020)
5. Frehse, G., Hamann, A., Quinton, S., Woehrle, M.: Formal analysis of timing effects on closed-loop properties of control software. In: RTSS. pp. 53–62 (2014)
6. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: CAV. LNCS 6806, Springer (2011)
7. Gabel, R.A., Roberts, R.A.: Signals and Linear Systems. John Wiley & Sons, 2nd edn. (1980)
8. Hobbs, C., Ghosh, B., Xu, S., Duggirala, P.S., Chakraborty, S.: Safety analysis of embedded controllers under implementation platform timing uncertainties. To appear, IEEE TCAD (2022)
9. Lunniss, W., Altmeyer, S., Davis, R.: Comparing FP and EDF accounting for cache related pre-emption delays. Leibniz Trans. on Emb. Sys. **1**(1), 01–1–01:24 (2014)
10. Maggio, M., Hamann, A., Mayer-John, E., Ziegenbein, D.: Control system stability under consecutive deadline misses. In: ECRTS. vol. 165, pp. 21:1–21:24 (2020)
11. Messner, W., Tilbury, D.: Control Tutorials for MATLAB and Simulink: A Web-based Approach. Addison-Wesley (1999)
12. Minaeva, A., Roy, D., Akesson, B., Hanzálek, Z., Chakraborty, S.: Control performance optimization for application integration. IEEE ToC. (2021)
13. Murphy, K.N.: In: Analysis of Robotic Vehicle Steering and Controller Delay (1994)
14. O'Kelly, M., Zheng, H., Karthik, D., Mangharam, R.: F1tenth: An evaluation environment for continuous control and reinforcement learning. In: NeurIPS (2019)
15. Pazzaglia, P., Pannocchi, L., Biondi, A., Natale, M.D.: Beyond the Weakly Hard Model: Cost of Deadline Misses. In: ECRTS. vol. 106, pp. 10:1–10:22 (2018)
16. Roy, D., Ghosh, S., Zhu, Q., Caccamo, M., Chakraborty, S.: Goodspread: Criticality-aware static sched. of CPS with multi-qos. In: RTSS. pp. 178–190 (2020)
17. Roy, D., Hobbs, C., Anderson, J.H., Caccamo, M., Chakraborty, S.: Timing debugging for cyber-physical systems. In: DATE. pp. 1893–1898 (2021)
18. Vreman, N., Cervin, A., Maggio, M.: Stability and performance analysis of control systems subject to deadline misses. In: ECRTS. vol. 196, pp. 15:1–15:23 (2021)
19. Vreman, N., Mandrioli, C., Anton, C.: Deadline-miss-adaptive controller implementation for real-time control systems. In: RTAS (2022)
20. Vreman, N., Mandrioli, C.: Evaluation of Burst Failure Robustness of Control Systems in the Fog. In: Workshop on Fog-IoT. OASIcs, Schloss Dagstuhl (2020)
21. Zhang, L., Lu, P., Kong, F., Chen, X., Sokolsky, O., Lee, I.: Real-time attack-recovery for CPS using linear-quadratic regulator. ACM TECS **20**(5s) (2021)