Quantitative Safety-Driven Co-Synthesis of Cyber-Physical System Implementations

Clara Hobbs*, Shengjie Xu*, Bineet Ghosh†, Enrico Fraccaroli*,
Parasara Sridhar Duggirala*, Samarjit Chakraborty*
*The University of North Carolina at Chapel Hill, USA †The University of Alabama, USA
Email: *{cghobbs, sxunique, enrifrac, psd, samarjit}@cs.unc.edu, †bineet@ua.edu

***Abstract*—Feedback controllers form the algorithmic core of many cyber-physical systems (CPSs). They are increasingly becoming computationally expensive and *efficiently implementing* them on resource-constrained platforms—such as those in the automotive domain—*while guaranteeing safety* is now an important challenge. Current workflows allow control strategies to be designed independently of the implementation environment and require control tasks to meet predetermined deadlines. Embedded systems engineers treat these control tasks as black boxes and focus on meeting *all* deadlines as the mechanism for ensuring safety. In this paper, we argue that deadlines are only a means to an end and should not be treated as "first-class citizens." Instead, the focus should be on high-level safety properties of relevance. Our main technical contribution is in automatic synthesis of safe CPS implementations: given a set of controllers to be implemented on a shared resource, along with their safety properties (a form of state space trajectory robustness), we synthesize an implementation that does not necessarily meet all task deadlines, but guarantees the safety specifications of all controllers.**

I. Introduction and Related Work

A. Background and motivation

The core functionality implemented by software in many cyber-physical systems consists of various feedback control loops [1]. In the automotive domain, these might be engine control, brake control, cruise control, and suspension & vibration control [2]. The traditional implementation workflow is to first design a controller, and then implement it as a software task that is scheduled to finish within a specified sampling period. Such a workflow ensures the separation of concerns, enabling control theorists and embedded systems engineers to only communicate via deadlines that must be met. Ensuring *safety* therefore equates to meeting *all* deadlines.

The rise in demand for implementing autonomous features into embedded systems such as in the automotive and robotics domains has resulted in two main developments: (i) a dramatic increase in the volume of software being deployed, and (ii) hardware architectures becoming more complex and heterogeneous to accommodate new features in software [3]. With increasing hardware and software complexity, tightly estimating the worst case execution time (WCET) of software tasks is virtually impossible [4], [5]. For WCET estimates to be safe, they are grossly overestimated. Meeting all task deadlines with such overestimated WCET values leads to pessimistic or infeasible implementations and scheduling control & signal processing tasks is also complex [6]. On the other hand, ignoring the WCET estimates and over-provisioning the hardware by scheduling an excessive number of tasks could be potentially hazardous, as some tasks might implement control mechanisms for safety critical aspects of the system. The real-time scheduler therefore faces a two-fold pressure—from the increased number of computationally expensive tasks to be scheduled, and the overly pessimistic WCET estimates. The correctness of any standard scheduler also relies on *safe* WCET estimates. These developments have made it increasingly challenging to engineer *safe* and *efficient* embedded software implementations.

B. Missing Deadlines on Purpose

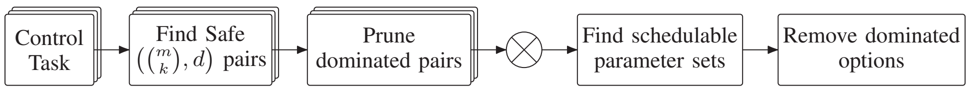
Our approach to resolving the above situation is to relax one of the fundamental tenets of real-time scheduling: that *every* task should meet its deadline. By allowing tasks to occasionally miss their deadlines, we can contain some of the pessimism stemming from the unavoidable WCET overestimation. Instead of pinning the "safety" of the system to the satisfaction of deadlines, we shift our focus to high-level safety properties of relevance, *viz.*, those that are specified in terms of the behavior of the closed-loop feedback system. These high-level safety properties can be satisfied even in the presence of certain deadline misses. Once such "allowable" deadline miss patterns are identified for each control task sharing a common resource, they can be used to suitably schedule the tasks.

This raises two key questions: *which deadlines may be missed?* and *how can we synthesize a scheduler that schedules only some of the tasks or allows the tasks to skip deadlines some of the time?* In answering these questions, we draw from two themes of work and merge them in a way that has not been done before. The first domain is formal methods & control theory, and the other domain is real-time scheduling.

C. Prior Work: Weakly Hard Constraints and Control Theory

In the formal methods domain, several recent papers [7]–[9] have investigated the effect of deadline misses on the dynamics of feedback controllers. Using principles of switched systems and reachability analysis, these works have demonstrated that feedback control loops are stable and safe for a variety of task deadline miss patterns. However, these works do not provide any insight for synthesizing schedulers when various control tasks share the same computational resource. Our own prior work [9], [10] has attempted to synthesize schedules for control tasks under deadline misses. But the underlying resource model was restrictive (purely time-triggered [11])

979-8-3503-6927-4/24/$31.00 ©2024 IEEE
DOI 10.1109/ICCPS61052.2024.00016



Control Task
Find Safe $\left(\binom{m}{k}, d\right)$ pairs
Prune dominated pairs
Find schedulable parameter sets
Remove dominated options

Fig. 1. The workflow of the proposed co-synthesis approach.

and did not exploit the variety of available algorithms on task scheduling under deadline misses, as we discuss below.

Work in the real-time scheduling literature has investigated the notion of *weakly hard constraints* [12]–[15]—where a task can miss some of its deadlines—for performing scheduling analysis. They formulate criteria to guarantee schedulability. However, they are agnostic to specific weakly hard constraints and provide no systematic means for deriving them. There also exists work on scheduling task sets to meet given weakly hard constraints [10], [16].

It is worth noting that deadline misses are prevalent in industrial practice. This is because WCET estimates are largely measurement based [17]–[20], rather than relying on static analysis [21]–[23], to avoid the overestimation outlined above. However, such deadline misses happen in an *ad hoc* manner and are compensated for by extensively testing software controller implementations and tweaking their parameters in a trial-and-error fashion. One of the primary motivations for studying weakly hard constraints is to provide a formal framework for real-time guarantees in the presence of such deadline misses stemming from practical estimates of WCETs. However, most existing work on weakly hard constraints is restricted to provide formal guarantees on task deadline misses but do not extend to guarantees on the system-level behavior of a dynamical system, as we provide in this paper.

D. *Our technical contributions*

Our main contribution lies in bridging the two above mentioned solutions: (i) synthesizing schedules for control tasks under deadline misses, and (ii) task scheduling to meet weakly hard constraints. We combine these to co-synthesize schedules for multiple controllers, while guaranteeing their individual safety specifications. By eliminating the constraint of meeting *all* deadlines, we are able to ensure efficient implementations and can schedule task sets with utilization greater than 100%, thereby addressing the pessimism of overestimating WCET values. Using standard design flows available today, such task sets could not be implementable without additional resources.

Synthesizing an implementation in our work amounts to deriving a schedule for multiple control tasks implemented on a single resource. This can be extended to a broader notion of implementation involving multiple computation and communication resources. The challenges involved in synthesizing such a schedule are threefold. First, missing some of the deadlines introduces non-determinism in the scheduler and the number of possible schedules increases combinatorially with the number of tasks. Second, for each state feedback controller, the deadline misses that can be tolerated are a function of its safety specification. Establishing this relationship between safety and deadline miss patterns is non-trivial. Here, we consider a quantitative notion of safety, *viz.,* the deviation of the closed-loop dynamics of the system from its ideal dynamics where tasks *always* meet their deadlines. This notion may be viewed as a form of *trajectory robustness*, where certain trajectories in the state space are considered to be safe and others not. While we only consider the deviation from an ideal trajectory, additional constraints or other notions of safety may be incorporated within our framework. Hence, in the rest of this paper we use the general term "quantitative notion of safety" to distinguish between safe and unsafe trajectories using their deviation from an *ideal* trajectory. A specific metric of *deviation* is introduced later in the paper. Our notion of safety has more practical relevance than merely ensuring *stability*, which a number of previous papers have studied in the context of task deadline misses [7], [24]–[26]. They also did not study schedule synthesis as we do here.

Finally, the safety specifications of one or more tasks could be interrelated, thus increasing the number of dependencies on the task scheduler. Often controllers corresponding to different sub-systems of the same system are implemented on a shared resource. In such cases, the *safety margin* of one controller may be adjusted when the margin of another one is changed. Such dependencies are difficult to uncover and typically each controller's safety margin is determined independently. We show how to uncover the trade-offs between the safety margins of the different controllers to ensure that they are implementable on a shared resource. This opens up new optimization opportunities and provides an insight into the design space that was not available before.

Our work leverages compositionality to combat the combinatorial explosion in the space of all possible schedulers. Our approach consists of three parts. First, for a given feedback control mechanism and its safety specification, we enumerate possible *weakly hard constraints* [12]–[15] on the control tasks that would preserve safety. Such constraints capture how many deadlines within a specified window of consecutive task periods need to be met, and are outlined in detail in Section III. Second, we prune the space of weakly hard constraints for each task by checking whether they satisfy the safety constraint for the task. Finally, we leverage existing schedulability tests for weakly hard constraints to synthesize schedulers for the control tasks. Since we consider quantitative safety properties, we also derive a Pareto front showing the trade-offs between the safety margins of the different tasks. We call our approach *co-synthesis* because the weakly hard constraints for all tasks sharing the implementation platform are determined jointly. This design flow is illustrated in Figure 1.

A new & flexible contract: At its core, this paper reimagines the contract between the control tasks and the schedulers from "*all* deadlines should be met," to "all the deadlines are *weakly hard*". This allows us the flexibility of swapping one scheduler to another, as long as the scheduler is able to schedule all the tasks with weakly hard constraints. Indeed, we show two different ways to synthesize the scheduler—

one is an online, dynamic priority scheduler, and the other is an offline, automata-theoretic one. In Section IV we highlight their relative (dis)advantages.

Related work in the broad area: We conclude this section by putting our work in the broader context of studies on the behavior of control systems under timing variations and deadline misses. This topic has generally been studied under the banner of *networked control systems* [27]–[29]. Here, the goal has been to characterize control performance in presence of (wireless) network packet drops and delays [30]. The drop or deadline miss characteristics in such cases are *given* and the goal has been to mitigate their effect by suitably designing the controller [31]. In contrast to such studies, in this paper *we determine* the drop or deadline miss patterns by appropriately designing a scheduler, and therefore solve a fundamentally different problem that is related to synthesizing controller implementations from control laws [32]. Problems closer to ours have been studied before, for instance, in [33], which seeks to optimize control performance while dropping jobs in overloaded processors. Their approach made use of an online arbiter to drop selected jobs, which unfortunately runs in exponential time. By contrast, our work has no such arbiter, and seeks to optimize performance in an offline fashion for efficiency. [34] seeks to solve a similar problem to ours, but focuses on stability instead of safety (as we do), and uses a different timing model. Further, their experiments test stability of switched systems by merely testing stability of individual modes, which is insufficient [35].

Paper organization: We provide background on linear control systems in Section II, on safety constraints in Section III, and weakly hard constraints and their representation in Section IV-A. The technical details of our proposed co-synthesis approach is provided in Section IV-B. We highlight the advantages of our approach by comparing with other scheduler synthesis frameworks in Section V. Finally, Section VI provides concluding remarks and future directions.

II. Control Theory Background

In this work, software tasks are assumed to implement state-feedback controllers for linear time-invariant (LTI) dynamical systems. These systems can be modeled in discrete time as

$$x[t+1] = Ax[t] + Bu[t], \tag{1}$$

where $A \in \mathbf{R}^{p\times p}$ is the *dynamics matrix*, $x[t] \in \mathbf{R}^p$ is the system *state* at time t, and $B \in \mathbf{R}^{p\times q}$ is the *input matrix*. The *control input* $u[t] \in \mathbf{R}^q$ is computed as $u[t] = Kx[t-1]$, giving a one-step delay between sampling the plant's state and applying the resulting control input. This is commonly referred to in the literature as the *logical execution time* (LET) paradigm. The matrix $K \in \mathbf{R}^{q\times p}$ is called the controller's *feedback gain*, and is assumed to be given in this work. A system of this form can be modeled by a single equation using an augmented state space $z[t] = [x[t]^T \;\; u_a[t-1]^T]^T$, by

$$z[t+1] = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} z[t] + \begin{bmatrix} 0 \\ I \end{bmatrix} u_a[t]. \tag{2}$$

This puts the delayed controller into the standard form of (1), allowing the feedback gain $K_a \in \mathbf{R}^{q\times(p+q)}$ to be computed by standard techniques such as pole placement or as a linear-quadratic regulator. This can be further simplified to

$$z[t+1] = \begin{bmatrix} A \quad B \\ K_a \end{bmatrix} z[t]. \tag{3}$$

Now we no longer assume that all deadlines will be met and therefore some control input updates will be missed, causing a deviation from the system's intended behavior.

III. Finding Safe Weakly Hard Constraints

While much of the real-time scheduling work considers *hard real-time* tasks, where all deadlines must be met to ensure a task's temporal correctness, this is overly restrictive in some cases. Indeed, many systems can tolerate some deadline misses, as long as too many of them do not occur consecutively, or within a bounded window. To characterize or model such hit/miss patterns, the *weakly hard real-time* model was proposed in [36], [37] and there has been a renewed interest in it recently [7], [12]–[14], [38]. This model allows some deadlines to be missed, as long as some are met according to a particular sliding window constraint. Several such types of constraints have been considered in the literature. In this work, we focus on *meet any* constraints, denoted $\binom{m}{k}$ (pronounced "meet any m out of k"), which require that out of any window of length k, at least m deadlines are met. A constraint of this form is equivalent to the m,k-firm model proposed in [36]. When considered over finite-length strings, these constraints make up regular languages, a property that we will use later when modeling system behavior under deadline misses.

Weakly hard constraints can be compared via the language of deadline hit/miss patterns they accept. We denote the language accepted by constraint $\binom{m}{k}$ as $\mathcal{L}(\binom{m}{k})$. A constraint $\binom{m}{k}$ is said to be *stronger* than $\binom{p}{q}$, denoted $\binom{m}{k} \preceq \binom{p}{q}$, if $\mathcal{L}(\binom{m}{k}) \subseteq \mathcal{L}(\binom{p}{q})$. An exact condition for this is given in [37], reproduced here for completeness.

Theorem 1 (Theorem 5 from [37])**.**

$$\binom{m}{k} \preceq \binom{p}{q} \iff p \leq \max\left\{ \left\lfloor \frac{q}{k} \right\rfloor m, q + \left\lceil \frac{q}{k} \right\rceil (m-k) \right\}.$$

In this section, we present a method for determining safe weakly hard constraints for a given control system. We first present our notion of safety in Section III-A, then discuss how safety can be proven in Section III-B. Finally, in Section III-C, we present a method for finding a set of safe constraints for a given system, keeping the necessary information for our co-synthesis approach to be discussed later.

A. Safety

In this section, we define the notion of safety considered in this work. We first introduce the behavior of a control system during deadline misses and then present the notion of system safety in presence of such misses. When a feedback control job misses its deadline, the engineer has the flexibility to either kill the job, or not change the schedule and let the job run its course, or not schedule more jobs of the same control task until the current job is finished. Similarly, the actuator that receives the actuation values from the control task can either give *zero* input when the job misses its deadline, or hold the previous actuation value, or perform some interpolation on the history of the actuation values. A thorough discussion on these strategies and their consequence to stability and safety have been presented in [7], [9]. In this work, we focus on the strategy of *killing* the job at its deadline.

For actuation, we consider two strategies: setting the control input to *zero*, and *holding* the previous control input when the job misses its deadline. In the former case, the dynamics of deadline miss are described by

$$z[t+1] = \begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} z[t], \tag{4}$$

and in the latter case, the control input is held via

$$z[t+1] = \begin{bmatrix} A & B \\ 0 & I \end{bmatrix} z[t]. \tag{5}$$

The various permutations of the scheduling and actuation strategies only affect the closed loop dynamics; as the readers will observe in Section IV, the algorithm for co-synthesis of scheduler for all the control tasks remains the same.

As the system evolves differently when deadlines are missed, this causes some *deviation* from the nominal behavior. For example, in an adaptive cruise control system, deviation may include a change in the vehicle's speed, or its lateral position within the lane. Small amounts of deviation may be acceptable, but it could clearly be unsafe for a vehicle to travel much too fast or to swerve out of its lane entirely. Thus, our goal is to place an upper bound on the amount of deviation that can occur under deadline misses.

To do this, we first define a *run* of a control task as a sequence of actions corresponding to each job of the task.

Definition 1. A *run* of a control task is a sequence $r = \langle r_1, r_2, \ldots, r_H \rangle$ of actions, where each action is either *hit* or *miss*. The *nominal run* r_{nom} is the run of all hits.

Given a run of a task, its *evolution* can be computed using (3) for each period when the deadline is met, and using (4) or (5) for each deadline miss.

Definition 2. The *evolution* of a run r is a sequence $evol(r) = \langle z[0], z[1], z[2], \ldots, z[H] \rangle$, where $z[0]$ is the initial state of the control system. Each subsequent state $z[t+1]$ is computed by (3) if $r_{t+1} = hit$, and by (4) or (5) if $r_{t+1} = miss$ under the zero or hold strategy, respectively.

From a run's evolution, we can compute its *deviation* from the nominal run as the maximum distance over time.

Definition 3. The *deviation* of a run r is the maximum distance over a metric $dis(\cdot,\cdot)$ between $evol(r)$ and the evolution of the nominal run $evol(r_{nom})$ at any time t, computed as

$$dev(r) = \max_{1 \le t \le H} dis\big(evol(r)[t], evol(r_{nom})[t]\big).$$

This notion of deviation can be easily extended to an analysis that considers an initial *set* $z[0]$ instead of one state by using, *e.g.*, Hausdorff distance between the reachable sets. Given a weakly hard constraint, the *maximum deviation* can be computed as the greatest deviation of any of its allowed runs.

Definition 4. The *maximum deviation* of a control system subject to a weakly hard constraint O_i is defined as

$$d_i = \max_{r \in \mathcal{L}(O_i)} dev(r).$$

We say that a system is *safe* under a weakly hard constraint O if the maximum deviation subject to O is at most a given upper bound. Intuitively, this means that under any run, the system's state can never veer too far from the evolution of the nominal run. Given that there have been several quantitative notions of safety such as robustness of Signal Temporal Logic [39], or probabilistic guarantees on safety [40], it is worth discussing why we selected the *deviation from the nominal behavior* for quantifying the notion of safety. Control design is a multi-objective optimization where the engineer must carefully balance the safety and performance requirements along with resource constraints. The engineer often does not have access to all the requirements expressed in a formal logic and a given design might implicitly satisfy several requirements. Further, given the continuity properties of closed loop systems, engineers expect the system behaviors to deviate from the *ideal behaviors* because of sensor and actuator noise and modeling uncertainties. We therefore build on this notion of continuity and quantify the safety of a control implementation as the deviation from the expected behavior. We demonstrate this notion of safety in Example 1.

Example 1 (Safety of an electric steering application)**.** We demonstrate the concept of safety, concerning deviations from the nominal trajectory, using an example of a permanent magnet synchronous motor used in an electric steering application. The system model we consider has two states and one control input. We use the parameters of the model and Tustin's method to discretize it and obtain the following system model [7]:

$$x[t+1] = \begin{bmatrix} 0.996 & 0.075 \\ -0.052 & 0.996 \end{bmatrix} x[t] + \begin{bmatrix} 0.100 & 0.003 \\ -0.003 & 0.083 \end{bmatrix} u[t];$$

where $x = [d \;\; q]^T$ are the state variables. An optimal feedback controller for this system under nominal timing behavior computed using LQR is:

$$u[t] = \begin{bmatrix} 0.9067 & 0.07384 & 0 & 0 \\ 0.01041 & 0.9685 & 0 & 0 \end{bmatrix} \begin{bmatrix} x[t-1] \\ u[t-1] \end{bmatrix}.$$

This controller is computed under nominal timing behavior, but it may suffer from deadline misses at runtime. In such cases, some deviation from the nominal trajectory is acceptable, as long as it falls within the safety margin. The nominal trajectory of the system, green in Figure 2, represents the ideal behavior (under no deadline misses). The cyan rectangles around the nominal trajectory represent the desired safety

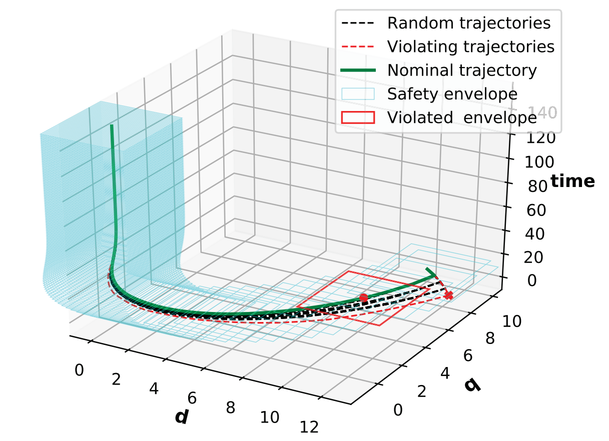


Fig. 2. Several trajectories of the electric steering example resulting from deadline misses in Example 1.

margin, where any deviation remaining within this region is considered safe. The safety margin at each time step represents the acceptable deviations in the states of the system. We computed several trajectories with deadline misses, shown in black and red in Figure 2. Although the system remains stable in the presence of deadline misses, some trajectories violate the safety property by going beyond the safety envelope (shown in red), while some remain within the safety margin (shown in black). The safety envelope highlighted in red is violated by one trajectory. The behavior would have been safe if the trajectory was within the red highlighted safety envelope, whereas it stretches outside to the point marked '×'. Note that the point is at a different time instant than the cyan rectangle it appears inside in the figure due to 2D projection.

Unfortunately, directly computing the maximum deviation as defined in Definition 4 is intractable, as the number of runs is exponential in the time horizon for a non-trivial constraint. We outline an efficient approach to bounding deviation next.

B. Proving Safety

To tackle the problem of efficiently determining if a controller is safe under a given weakly hard constraint, [9] recently proposed a reachability-based algorithm for the switched dynamics resulting from deadline misses. We outline the approach in this section, and refer the reader to that work for a more in-depth discussion.

The method begins by first modeling weakly hard constraints and the resulting plant dynamics as a finite-state automaton. An example of such an automaton is shown in Figure 3. Each location in the automaton corresponds to some past pattern of deadline hits and misses, and a transition is taken on each hit or miss. In the example, each location ℓ_i corresponds to having just missed i deadlines in a row. As we assume that the scheduler satisfies the weakly hard constraint, the final location ℓ_N has no outgoing transition on a deadline miss. On a transition, a matrix representing the dynamics and feedback gain is produced as output, thus enabling the computation of a run's evolution as in Definition 2. In the example, the matrix from (3) is denoted as $A_{\mathcal{H}}$, and the matrix from (4) or (5) is denoted as $A_{\mathcal{M}}$.

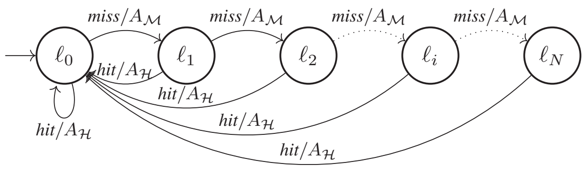

$$\begin{tikzcd}
\arrow[r] & \ell_0 \arrow[r, bend left, "miss/A_{\mathcal{M}}"] \arrow[loop below, "hit/A_{\mathcal{H}}"] & \ell_1 \arrow[r, bend left, "miss/A_{\mathcal{M}}"] \arrow[l, "hit/A_{\mathcal{H}}"] & \ell_2 \arrow[r, dotted, bend left, "miss/A_{\mathcal{M}}"] \arrow[ll, bend left, "hit/A_{\mathcal{H}}"] & \ell_i \arrow[r, dotted, bend left, "miss/A_{\mathcal{M}}"] \arrow[lll, bend left, "hit/A_{\mathcal{H}}"] & \ell_N \arrow[llll, bend left, "hit/A_{\mathcal{H}}"]
\end{tikzcd}$$

Fig. 3. Automaton capturing the $\binom{1}{N+1}$ weakly hard constraint.

Given such an automaton, the first iteration of the algorithm proceeds as follows. Starting at the initial location from plant state $z[0]$, the algorithm computes all reachable states up to some short time bound b. It then computes the axis-aligned bounding box of all plant states that are reachable in each location. An example is shown in Figure 6 in the appendix, where the initial plant state is $[1.0\ 1.0]^T$ at $t = 0$, and the constraint used is $\binom{1}{4}$. The plant's possible evolutions are simulated for 7 sampling periods, at which time, axis-aligned bounding boxes are taken for each of the four automaton locations.

In each following iteration, the algorithm computes the reachable sets from each of these bounding boxes, stating from each corresponding automaton location. For linear systems, this can be done by computing the evolution of each corner of the bounding box. The boxes for each location are combined into a single box, setting up similar conditions for each subsequent iteration. Running $\lceil \frac{H}{b} \rceil$ iterations, an overapproximation of all runs of the system is computed efficiently for the time horizon H. Given the reachable sets until time H, the deviation is computed as the maximum distance between any point in these sets and the nominal run's evolution.

C. Finding Safe Constraints

Given this reachability approach to finding an upper bound on the maximum deviation, it is straightforward to find safe weakly hard constraints for a given plant model and controller. One could simply compute upper bounds on deviation for all $\binom{m}{k}$ constraints up to a maximum window size k, then check which ones satisfy the system's safety constraint. However, this approach is inefficient, as it ignores that some weakly hard constraints dominate others as shown in Theorem 1. If for constraints O and P, $O \preceq P$, and O is unsafe, then it is clear that P is also unsafe. Similarly, if P is safe, then O must also be safe, as all its runs are contained within P.

From these observations, [10] proposes an efficient method for finding safe $\binom{m}{k}$ weakly hard constraints for a given control system model. This approach iterates over the window size k up to a given maximum, finding the smallest m for which the system is safe at each window size. Once such a safe constraint is found, Theorem 1 implies that all larger m values for the same window size are also safe. Therefore, the iteration increments the window size to try a harder constraint, without changing m. In this way, one can find the safety of a quadratic number of constraints in a linear number of calls to the algorithm from Section III-B.

This approach suffices if one is only interested in finding which constraints are safe. However, in this work, we propose to consider safety as a constraint to a multi-objective optimization problem, discussed later in Section IV-B. The objective of this optimization is to minimize deviation for all tasks in the system. As such, we must also find the actual deviation bounds achieved for each safe constraint.

Our approach to this checks weakly hard constraints $\binom{m}{k}$ as follows. We iterate over the number of met deadlines m, checking each window size k from $m+1$ to the given maximum window size, and storing the resulting deviation bounds. When a constraint $\binom{m}{k}$ is found to be unsafe, i.e., the deviation bound computed exceeds the safe limit, we move on to the next number of met deadlines, starting the window size iteration over. In this way, we only check one *unsafe* constraint for each m, but we still check every *safe* constraint to determine its resulting deviation bound.

This approach provides a means of determining safe constraints for each control system under consideration, as well as the resulting deviation bounds for each. This acts as the first step to our co-synthesis approach for cyber-physical system implementations. To use these constraints, however, we must also have a way to schedule tasks so that the weakly hard constraints are satisfied. We examine two such approaches below in Section IV-A.

IV. Co-Synthesis Approach

In this section, we give a brief survey of real-time schedulers for tasks with weakly hard constraints in Section IV-A. We then discuss how we use these schedulers in our co-synthesis approach for implementing multiple control tasks on a shared processor in Section IV-B.

A. Scheduling with Weakly Hard Constraints

A number of approaches have been proposed in the literature for scheduling periodic task systems with weakly hard constraints. These can be broadly classified as either offline schedulers, which generate a fixed time-driven schedule before runtime; or online schedulers, which react to events such as job releases and completions to schedule the task system at runtime. We consider both approaches in this work via a representative example of each: an online job-class-level scheduler [16], and an offline automaton-based scheduler [10] that has been used for safe scheduling of control systems. The operation of these are detailed in this section, and we note that *any* weakly hard scheduler could be used with our co-synthesis approach, not only these two. Before presenting these, we introduce our real-time task model in the next section.

1) Task Model: We consider the scheduling of a set of periodic real-time tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ on a uniprocessor. Each task $\tau_i = (T_i, D_i, C_i, O_i)$, where T_i is the period or interarrival time, D_i is the relative deadline, C_i is the worst-case execution time, and O_i is a weakly hard constraint for the task. The constraint O_i represents a pattern of deadlines that must be met for the task's temporal correctness.

In real-time scheduling, it is often useful to consider a task's *utilization*, representing the long-term average processor share required by each task to ensure its safe execution. When weakly hard constraints are considered, however, the notion of utilization becomes less clear, as not all instances of a task are required to complete, or even begin, execution. Therefore, we consider two types of utilization.

Definition 5. The *maximum utilization* of a task τ_i is given by $\overline{U}_i = C_i/T_i$.

Definition 6. The *minimum utilization* of a task τ_i is given by $\underline{U}_i = C_i/T_i \cdot m_i/k_i$, where $O_i = \binom{m_i}{k_i}$.

Both types of utilization may be computed for a task system τ by summing the utilization for all tasks in τ, and are denoted with no subscript. In this work, we assume that the maximum utilization $\overline{U}$ of the task system is greater than 1, meaning that the task system cannot be scheduled without some deadline misses. For a task system to be schedulable, it is necessary, but not sufficient, that its minimum utilization $\underline{U} \leq 1$. The weakly hard constraints of non-control tasks are given, and are not adjusted by our approach. However, we assume that changes to the weakly hard constraints of control tasks can be made as long as their safety is still guaranteed.

2) Job-class-level scheduling: To provide an efficient method of scheduling periodic weakly hard tasks, Choi, Kim, and Zhu proposed a dynamic-priority online scheduling approach [16]. Their algorithm divides the jobs of each task into *job-classes* on the basis of the previous number of consecutive deadline hits. Each job-class of each task is assigned a fixed priority using a heuristic. Given job-class priorities and the parameters of each task, a schedulability test is used to determine if any violations of the weakly hard constraints are possible during online scheduling. If no such violations are possible, the task system is deemed schedulable. A description of the scheduler's operation is included in Appendix A.

3) Automata-based scheduling: To provide a baseline in our experimental evaluation (Section V), we consider an automata-based scheduling approach for weakly hard tasks, which has been used in the context of guaranteeing safety for multiple controllers [10]. This scheduler is optimal, in that it will always find a feasible schedule for the task system if one exists. However, it can only be used in the limited context of synchronous weakly hard tasks with equal periods.

The automaton-based scheduler divides time into *slots* of one sampling period. The key assumption is that in each slot, a fixed number of jobs j can be scheduled, less than the number of tasks in τ. Thus, some deadlines must be missed. The scheduler operates offline, creating a time-driven schedule. In each slot, at most j tasks release jobs. The scheduler's operation is described in more detail in Appendix B.

B. Co-Optimizing Deviation for Multiple Controllers

The work in [10] provides a method of guaranteeing safety for a set of control tasks implemented on a shared platform that lacks sufficient computational resources to run all controllers as designed. However, when several safe schedules are found, it offers no guidance to system designers on which schedule to use. For example, consider a system with two control tasks, τ_1 and τ_2. It may be that these are both safe when run with the weakly hard constraint $\binom{1}{3}$. By Theorem 1, they must also be safe under $\binom{2}{3}$, whereby more deadlines must be met. As meeting more deadlines generally causes lower deviation from the nominal behavior, this may be desirable, but the system may not have the resources to run *both* tasks with this stronger constraint. While similar scenarios may easily arise in practice, prior work has not considered how to determine, e.g., which task to run with the $\binom{2}{3}$ constraint in our scenario.

In this section, we introduce a novel co-optimization approach that seeks to answer this question by minimizing the deviation of all tasks via a multi-objective optimization formulation. We assume that each control task controls an independent plant, that is, the execution of one task does not impact the state of another. Our formulation takes the weakly hard constraints O_i for each control task τ_i as variables, and has the minimization of the vector of deviation upper bounds as its objective. The optimization problem is as follows:

$$\min_{O_i}\langle d_1, d_2, \ldots, d_n\rangle \quad \text{s.t.}(\forall i : d_i \leq \overline{d}_i) \wedge (\tau \text{ is schedulable})$$

A solution S corresponds to an assignment of weakly hard constraints O_i to each control task τ_i. Such a solution results in a deviation upper-bound d_i for each control task, each satisfying a safe deviation $\overline{d}_i$. A solution S is said to be *dominated* by another solution S' if for all i, $d_i \geq d'_i$. As this is a multi-objective optimization problem, there is generally not a single solution, but a set of non-dominated, Pareto-optimal solutions referred to as the *Pareto front*.

Due to the dynamics of a control system under deadline hits and misses, this formulation presents a non-linear, non-convex optimization problem, not amenable to standard tools such as mixed-integer linear program solvers. As such, we propose a custom solution that can efficiently solve the problem through pruning via the dominance of weakly hard constraints. We explain this approach in Example 2 by way of a simple example to demonstrate its usage.

Example 2. To illustrate the operation of our optimization approach, we present a numerical example using two control tasks and one non-control task. The dynamics of control task τ_1, discretized at a period of 20 ms, are given by

$$z[t+1] = \begin{bmatrix} 1 & 0.12 & 0.024 \\ 0 & 1 & 0.4 \\ 0 & 0 & 0 \end{bmatrix} z[t] + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_a[t],$$
$$u_a[t] = \begin{bmatrix} 0.584 & 0.901 & 0.347 \end{bmatrix} z[t].$$

The dynamics of the second control task τ_2, discretized at a period of 10 ms, are given by

$$z[t+1] = \begin{bmatrix} 0.951 & 0.00971 & 0.000049 \\ 0.000097 & 0.990 & 0.00995 \\ 0 & 0 & 0 \end{bmatrix} z[t] + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u_a[t],$$
$$u_a[t] = \begin{bmatrix} 0.00706 & 0.227 & 0.00228 \end{bmatrix} z[t].$$

Control task τ_1 has period $T_1 = 20$ ms, WCET $C_1 = 6$ ms, and implicit deadline (equal to its period). Control task τ_2 has period $T_2 = 10$ ms, WCET $C_2 = 3$ ms, and implicit deadline. Finally, non-control task τ_3 has period $T_3 = 130$ ms, WCET $C_3 = 70$ ms, and implicit deadline. The weakly hard constraint of τ_3 is given by $O_3 = \binom{2}{5}$.

For this task system, $\overline{U} \approx 1.138$, indicating that this system cannot be scheduled without some deadline misses. Our goal is to find weakly hard constraints for tasks τ_1 and τ_2 that guarantee their safety, and the task system's schedulability, while keeping their deviation as low as possible. Both controllers are assumed to start at the state $z[0] = [10 \;\; 10 \;\; 0]^T$. Task τ_1 is considered safe if its first state variable $z_1[t]$ remains within 5 units of the nominal evolution, and task τ_2 is safe if its first state variable remains within 0.01 units of nominal.

To begin the optimization procedure, listed in Algorithm 1, we assume that upper bounds to deviation are available for all safe weakly hard constraints for each controller. We are implicitly assuming that the safety requirements of each controller are independent. While in general there could be dependencies among safety requirements between controllers, handling such requirements is beyond the scope of this work. Recall that these bounds can be obtained as described in Section III-C. Recalling Theorem 1, some weakly hard constraints can be found to be stronger than others, meaning that the language of strings they accept is contained within another's. Thus, a weaker constraint allows strictly more flexibility for scheduling than a stronger one.

Our first step in optimization is to use this relation to *prune* weakly hard constraints that will never produce optimal results. Given two constraints O and O' and their respective deviation bounds d and d', if $O \preceq O'$, and $d = d'$, then the pair (O, d) is pruned on line 6. Intuitively, this is safe because (O, d) is no better for scheduling or control performance than (O', d').

Example 2 (continued)**.** We first check the safety of all miss-any weakly hard constraints up to a maximum window size of 6. For each safe constraint, the deviation bound is shown in Table I. Constraints that could not be verified as safe are indicated by an X. The bounds on each diagonal are all equal for τ_1, and are close to equal for τ_2. Because of this, the pruning procedure is able to remove all constraints for τ_1 except $\binom{1}{2}$ and $\binom{1}{3}$, but does not remove any constraints for τ_2.

After pruning, we examine the Cartesian product of the safe constraints for each task in the loop on line 8. We iterate through these sets of constraints in the lexicographical order of the deviation vectors they produce. For each set of constraints considered, we test the resulting task system's schedulability. For efficiency, we first test the necessary condition that $\underline{U} \leq 1$ on line 10, rejecting any combinations for which this does not hold. We also avoid running the schedulability test if the resulting deviation vector is dominated by a candidate solution we have already found, as we then know that this set of constraints will never be part of the Pareto front. Searching

TABLE I
SAFE DEVIATION BOUNDS $\binom{m}{k}$ FOR THE CONTROL TASKS τ_1 (TOP) AND τ_2 (BOTTOM, ALL VALUES $\times 10^{-3}$) IN EXAMPLE 2

	m				
k	1	2	3	4	5
2	1.6714				
3	3.3944	1.6714			
4	✗	3.3944	1.6714		
5	✗	✗	3.3944	1.6714	
6	✗	✗	✗	3.3944	1.6714

	m				
k	1	2	3	4	5
2	2.9805				
3	5.9281	2.9800			
4	8.8443	5.9275	2.9799		
5	✗	8.8435	5.9272	2.9798	
6	✗	✗	8.8430	5.9269	2.9796

in lexicographical order helps to increase the number of constraint sets we can skip in this way. After this pruning, we run a schedulability test for the scheduling algorithm under consideration on line 12. If this test passes, then the solution is added as a candidate for the Pareto front. After all combinations from the Cartesian product have been tested, we eliminate non-optimal solutions by the dominance of deviation vectors to find the Pareto front on line 15.

```
1  Function ScheduleOptimization($\tau$, $\mathbf{O}_1$, $\mathbf{O}_2$, ..., $\mathbf{O}_n$)
       input : Task system $\tau$, mapping of safe constraints to deviation
               bounds $\mathbf{O}_i$ for each control task $\tau_i$
       output: Mapping from optimal task systems to vectors of
               deviation bounds
2      foreach control task $\tau_i \in \tau$ do
3          foreach constraint, deviation pair $(O, d) \in \mathbf{O}_i$ do
4              foreach constraint, deviation
                 pair $(O', d') \neq (O, d) \in \mathbf{O}_i$ do
5                  if $O \preceq O'$ and $d = d'$ then
6                      Delete $(O, d)$ from $\mathbf{O}_i$ ;
7      candidates $\leftarrow \varnothing$;
8      foreach $(\langle O_1, O_2, \ldots, O_n \rangle, \langle d_1, d_2, \ldots, d_n \rangle) \in$
         $\mathbf{O}_1 \times \mathbf{O}_2 \times \cdots \times \mathbf{O}_n$, in lexicographical order of deviation
         vectors do
9          $\tau^O \leftarrow \tau$ with constraints $\langle O_1, O_2, \ldots, O_n \rangle$;
10         if $\underline{U}^O > 1$ or $\langle d_1, d_2, \ldots, d_n \rangle$ is dominated by a
             candidate then
11             continue;
12         if $\tau^O$ is schedulable then
13             Add $(\tau^O, \langle d_1, d_2, \ldots, d_n \rangle)$ to candidates;
14     $P \leftarrow \varnothing$;
15     foreach candidate solution do
16         if not dominated by another solution then
17             Add solution to $P$;
18     return $P$;
```

Algorithm 1: The proposed co-optimization approach for minimizing deviation.

Example 2 (continued)**.** All the schedulable solutions for our example are plotted in Figure 4. Each point represents the deviation bounds for a safe set of weakly hard constraints; d_1 is shown by the horizontal axis, and d_2 on the vertical axis. The optimal solutions are highlighted in blue and annotated with the weakly hard constraints for tasks τ_1 and τ_2, and the dominated solutions are shown in gray.

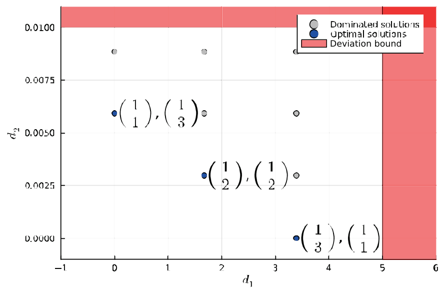


Fig. 4. Solutions for Example 2. Optimal points are marked with O_1, O_2.

V. EVALUATION

In this section, we present experimental results from a case study conducted using five control tasks from the automotive domain. We examine controllers for these tasks first having equal periods, then with redesigned controllers for unequal sampling periods. In both cases, we are able to find schedules using our method that minimize the deviation of the control systems, offering tradeoffs to system designers.

To test our methods, we produced an implementation in the Julia [41] programming language. This implementation includes our co-synthesis approach, as well as analysis for the two scheduling algorithms discussed in Section IV-A. All experiments were performed on an Intel E5-2680 processor with 256 GB of RAM, running Red Hat Enterprise Linux 7.4. We consider the execution of controllers for five dynamical systems from the automotive domain. **The parameters of these systems in continuous time are listed in Appendix C.**

A. Equal Period Case Study

For our first case study, we consider the systems from Appendix C, all sampled at a uniform period of 20 ms, matching the case study in [10]. The control tasks for the systems from Sections C1–C5 are denoted by τ_1–τ_5, respectively. For simplicity, we consider no non-control tasks in this case study. The WCETs of τ_1–τ_5 are 5 ms, 6 ms, 3 ms, 11 ms, and 9 ms, respectively. The maximum utilization is $\overline{U} = 1.7$, meaning the system cannot be scheduled without missing some deadlines. The RC Network model is considered safe if the difference between its state variables never exceeds 1.4 V from nominal. Safety for all other systems is determined by the difference in the first state variable only. The safe bound for the F1tenth Steering model is 12.0; for the DC Motor model, 3.5; for the Car Suspension model, 9.4; and for the Cruise Control model, 5.3. The initial state considered in all systems is $x[0] = [10\ \ 10\ \ \mathbf{0}]^T$.

We first determine safe constraints for this case study. These are given in Table III in the appendix, where unsafe constraints are denoted by ✗ and safe but dominated constraints are denoted by ✓. As is clearly seen, a number of safe, non-dominated constraints are found for each system, leaving a wide range of tradeoffs to be explored by co-synthesis.

We next attempt to find feasible schedules under combinations of these constraints using the job-class-level scheduler described in Section IV-A2. Unfortunately, the case of all

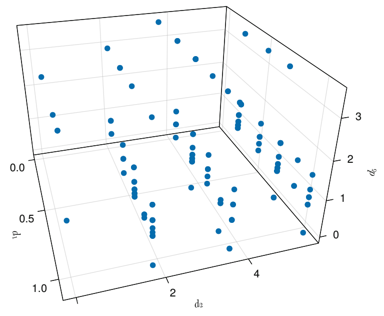


Fig. 5. Pareto front for equal periods, showing deviation for τ_1, τ_2, and τ_5.

periods being equal is pathological for this algorithm's schedulability analysis, and reduces to testing $\overline{U} \leq 1$. Since this is not the case here, no schedules are found to be feasible for this scheduler. We thus move on to the automata-based scheduler described in Section IV-A3. Given the WCETs of each task, this algorithm schedules a maximum of two tasks per 20 ms sampling period. Despite this limitation, the scheduler is quite robust for equal-period tasks, finding a total of 133 Pareto-optimal sets of weakly hard constraints for this task set. A projection of these into three dimensions for tasks τ_1, τ_2, and τ_5 is shown in Figure 5. The points that appear to be dominated in this plot achieve lower deviation bounds for tasks τ_3 or τ_4, and therefore are still Pareto optimal.

B. *Varying Period Case Study*

For our second case study, we consider the same systems from our experimental setup, but no longer sampled at a uniform period. Instead, we now take the periods for τ_1–τ_5 to be 20 ms, 20 ms, 10 ms, 100 ms, and 50 ms, respectively, still with all deadlines equal to periods. The WCETs are 6 ms, 7 ms, 3 ms, 15 ms, and 11 ms, respectively. Otherwise, the experimental setup is the same as in Section V-A. In this case, the maximum utilization $\overline{U} = 1.32$, again indicating that the system cannot be scheduled without some deadline misses.

As before, we begin by finding safe deviation bounds for each of the controllers in this case study. The results are given in Table IV in the appendix. As before, a range of tradeoffs is available for each task. It is worth noting that the deviation values for the DC Motor model differ slightly along each diagonal, so that no pruning was possible.

Because the periods are not equal, we can no longer use the automata-based scheduler. However, the job-class-level scheduler can be used for this case study. Using the schedulability test given in [16], we find only 7 optimal solutions, out of a total of 19,200 possible combinations of weakly hard constraints, 2,439 of which are schedulable. The optimal solutions are listed in Table II. The small size of the Pareto front highlights the need for a methodology to find the optimal schedules, as most feasible task systems are not optimal in terms of the deviation bounds they provide.

C. *Discussion*

The case studies we have provided give useful examples of the efficacy of our co-synthesis approach. Since the systems both have $\overline{U} > 1$, they could not even be scheduled without missing some deadlines. In both cases, we are able to find safe schedules for the task system that offer a tradeoff between the control performance of the various controllers. Furthermore, these schedules are optimal in the deviation bounds produced.

TABLE II
PARETO-OPTIMAL SOLUTIONS FOUND FOR SECTION V-B

O_1	O_2	O_3	O_4	O_5	d_1	d_2	d_3	d_4	d_5
$\binom{1}{2}$	$\binom{1}{1}$	$\binom{1}{4}$	$\binom{1}{1}$	$\binom{1}{1}$	0.319	0.0	0.008	0.0	0.0
$\binom{1}{3}$	$\binom{1}{1}$	$\binom{1}{2}$	$\binom{1}{1}$	$\binom{1}{1}$	0.577	0.0	0.003	0.0	0.0
$\binom{1}{1}$	$\binom{1}{3}$	$\binom{1}{2}$	$\binom{1}{1}$	$\binom{1}{1}$	0.0	3.641	0.003	0.0	0.0
$\binom{1}{2}$	$\binom{1}{3}$	$\binom{1}{1}$	$\binom{1}{1}$	$\binom{1}{1}$	0.319	3.641	0.0	0.0	0.0
$\binom{1}{3}$	$\binom{1}{2}$	$\binom{1}{1}$	$\binom{1}{1}$	$\binom{1}{1}$	0.577	1.786	0.0	0.0	0.0
$\binom{1}{2}$	$\binom{1}{2}$	$\binom{1}{2}$	$\binom{1}{1}$	$\binom{1}{1}$	0.319	1.786	0.003	0.0	0.0
$\binom{1}{1}$	$\binom{1}{2}$	$\binom{1}{3}$	$\binom{1}{1}$	$\binom{1}{1}$	0.0	1.786	0.006	0.0	0.0

Each case study makes use of a different scheduling algorithm, illustrating that our proposed approach can be used with *any* scheduler supporting tasks with $\binom{m}{k}$ constraints. This even extends to fundamentally different approaches to dealing with these constraints, such as online vs. offline scheduling. With an offline scheduler, as in Section V-A, the system's utilization is effectively *forced* to be lower by skipping certain jobs entirely. Even so, good control performance can still be achieved by giving enough resources to each control task. Conversely, online schedulers, such as the one considered in Section V-B, optimistically try to run every job of every task. Because most jobs require less than the WCET, this may be successful, resulting in better control performance for all plants. However, if enough tasks do require execution times close to their worst case, the weakly hard guarantees still hold, and consequently, our deviation bounds hold as well.

VI. Concluding Remarks

We have demonstrated how quantitative safety properties can be guaranteed for a number of control tasks implemented on a shared platform, while optimizing for their control performance and offering tradeoffs between them. This quantitative notion of safety is guaranteed despite allowing some deadline misses for the control tasks, contrary to the typical hard real-time approach. By relaxing the requirement that all deadlines must be met, and instead focusing on a *weakly hard* setting in which some deadlines may be missed as long as a minimum level of service is guaranteed, we can keep the deviation of all control systems to a minimum. This is achieved by a co-synthesis approach, whereby safe weakly hard constraints are determined for each task, and then combinations are tried to find those that are Pareto-optimal. This offers a set of choices to control designers, while accommodating non-control tasks with weakly- or strongly-hard real-time requirements.

For future work, we plan to explore the use of other measures of control performance beyond deviation in our co-synthesis scheme. Using common metrics such as change in rise time and settling time, it may be easier for control designers to interpret the change in performance of controllers subjected to occasional deadline misses. It may also prove useful to add methods of reducing the dimensionality of the co-synthesis problem. This could be achieved, *e.g.*, by grouping subsets of the tasks together, and minimizing a function of the deviations for each group. In this way, related high-level functionality of the system could be optimized, while simplifying the presentation of the Pareto front to designers.

Acknowledgments: This work was partially supported by the NSF grant# 2038960. The authors also thank the anonymous reviewers and the shepherd, Ivan Ruchkin, for their constructive suggestions to improve the paper.

REFERENCES

[1] S. Chakraborty *et al.*, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Des. Test*, vol. 33, no. 4, pp. 92–108, 2016.

[2] W. Chang and S. Chakraborty, "Resource-aware automotive control systems design: A cyber-physical systems approach," *Found. Trends Electron. Des. Autom.*, vol. 10, no. 4, pp. 249–369, 2016.

[3] G. Georgakos *et al.*, "Reliability challenges for electric vehicles: from devices to architecture and systems software," in *50th Annual Design Automation Conference (DAC)*, 2013.

[4] R. Wilhelm, "Determining reliable and precise execution time bounds of real-time software," *IT Professional*, vol. 22, no. 3, pp. 64–69, 2020.

[5] C. Berg, J. Engblom, and R. Wilhelm, "Requirements for and design of a processor with predictable timing," in *Perspectives Workshop: Design of Systems with Predictable Behaviour*. IBFI, Schloss Dagstuhl, 2004.

[6] S. Chakraborty and L. Thiele, "A new task model for streaming applications and its schedulability analysis," in *Design, Automation and Test in Europe Conference and Exposition (DATE)*, 2005.

[7] M. Maggio *et al.*, "Control-System Stability Under Consecutive Deadline Misses Constraints," in *32nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2020, pp. 21:1–21:24.

[8] P. Pazzaglia *et al.*, "DMAC: Deadline-Miss-Aware Control," in *31st Euromicro Conference on Real-Time Systems (ECRTS)*, 2019.

[9] C. Hobbs, B. Ghosh, S. Xu, P. S. Duggirala, and S. Chakraborty, "Safety analysis of embedded controllers under implementation platform timing uncertainties," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4016–4027, 2022.

[10] S. Xu, B. Ghosh, C. Hobbs, P. Thiagarajan, and S. Chakraborty, "Safety-aware flexible schedule synthesis for cyber-physical systems using weakly-hard constraints," in *28th ASP-DAC*, 2023.

[11] D. Goswami *et al.*, "Time-triggered implementations of mixed-criticality automotive software," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.

[12] N. Vreman, R. Pates, and M. Maggio, "Weaklyhard.jl: Scalable analysis of weakly-hard constraints," in *RTAS*, 2022.

[13] P. Pazzaglia, L. Pannocchi, A. Biondi, and M. D. Natale, "Beyond the weakly hard model: Measuring the performance cost of deadline misses," in *ECRTS*, 2018.

[14] Z. Hammadeh *et al.*, "Bounding deadline misses in weakly-hard real-time systems with task dependencies," in *DATE*, 2017.

[15] Y. Sun and M. D. Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 5s, 2017.

[16] H. Choi, H. Kim, and Q. Zhu, "Toward practical weakly hard real-time systems: A job-class-level scheduling approach," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6692–6708, 2021.

[17] J. P. Hansen, S. A. Hissam, and G. A. Moreno, "Statistical-based WCET estimation and validation," in *9th Intl. Workshop on Worst-Case Execution Time Analysis (WCET)*, ser. OASIcs, vol. 10. Schloss Dagstuhl, Germany, 2009.

[18] A. Horga, S. Chattopadhyay, P. Eles, and Z. Peng, "Measurement based execution time analysis of GPGPU programs via SE+GA," in *21st Euromicro Conference on Digital System Design (DSD)*, 2018.

[19] H. Shah *et al.*, "Measurement based WCET analysis for multi-core architectures," in *22nd International Conference on Real-Time Networks and Systems (RTNS)*, 2014.

[20] I. Wenzel *et al.*, "Measurement-based worst-case execution time analysis," in *3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS)*, 2005.

[21] P. Axer *et al.*, "Building timing predictable embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4, pp. 82:1–82:37, 2014.

[22] R. Wilhelm, "Why AI + ILP is good for WCET, but MC is not, nor ILP alone," in *5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2004.

[23] L. Thiele and R. Wilhelm, "Design for timing predictability," *Real-Time Systems*, vol. 28, no. 2-3, pp. 157–177, 2004.

[24] S. Linsenmayer and F. Allgöwer, "Stabilization of networked control systems with weakly hard real-time dropout description," in *CDC*, 2017.

[25] A. Aminifar, E. Bini, P. Eles, and Z. Peng, "Bandwidth-efficient controller-server co-design with stability guarantees," in *Design, Automation & Test in Europe (DATE)*, 2014.

[26] A. Aminifar, P. Eles, Z. Peng, A. Cervin, and K. Årzén, "Control-quality-driven design of embedded control systems with stability guarantees," *IEEE Design & Test*, vol. 35, no. 4, pp. 38–46, 2018.

[27] W. Zhang, M. S. Branicky, and S. M. Phillips, "Stability of networked control systems," *IEEE Control Systems Magazine*, vol. 21, no. 1, pp. 84–99, 2001.

[28] M. C. F. Donkers *et al.*, "Stability analysis of stochastic networked control systems," *Automatica*, vol. 48, no. 5, pp. 917–925, 2012.

[29] D. Soudbakhsh, L. T. X. Phan, A. M. Annaswamy, and O. Sokolsky, "Co-design of arbitrated network control systems with overrun strategies," *IEEE Trans. Control. Netw. Syst.*, vol. 5, no. 1, pp. 128–141, 2018.

[30] D. Goswami *et al.*, "Characterizing feedback signal drop patterns in formal verification of networked control systems," in *IEEE International Symposium on Computer-Aided Control System Design (CACSD)*, 2013.

[31] M. B. G. Cloosterman *et al.*, "Controller synthesis for networked control systems," *Automatica*, vol. 46, no. 10, pp. 1584–1594, 2010.

[32] E. S. Kim, M. Arcak, and S. A. Seshia, "Flexible computational pipelines for robust abstraction-based control synthesis," in *31st International Conference on Computer Aided Verification (CAV) Part I*, ser. Lecture Notes in Computer Science, vol. 11561, 2019.

[33] T. Yoshimoto and T. Ushio, "Optimal arbitration of control tasks by job skipping in cyber-physical systems," in *IEEE/ACM Second International Conference on Cyber-Physical Systems (ICCPS)*, 2011.

[34] H. S. Chwa, K. G. Shin, and J. Lee, "Closing the gap between stability and schedulability: A new task model for cyber-physical systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018, pp. 327–337.

[35] Z. Sun and S. S. Ge, *Stability theory of switched dynamical systems*. Springer, 2011.

[36] M. Hamdaoui and P. Ramanathan, "A dynamic priority assignment technique for streams with (m, k)-firm deadlines," *IEEE Transactions on Computers*, vol. 44, no. 12, 1995.

[37] G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," *IEEE transactions on Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[38] C. Huang, K. Chang, C. Lin, and Q. Zhu, "SAW: A tool for safety analysis of weakly-hard systems," in *32nd International Conference on Computer Aided Verification (CAV)*, 2020.

[39] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, 2009.

[40] P. Zuliani, A. Platzer, and E. M. Clarke, "Bayesian statistical model checking with application to simulink/stateflow verification," in *13th ACM international conference on Hybrid systems: Computation and Control (HSCC)*, 2010.

[41] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017.

[42] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *RTSS*, 1990.

[43] R. A. Gabel and R. A. Roberts, *Signals and Linear Systems*, 2nd ed. John Wiley & Sons, 1980.

[44] O'Kelly *et al.*, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," *Proceedings of Machine Learning Research*, vol. 123, 2020.

[45] W. C. Messner and D. M. Tilbury, "Control tutorials for MATLAB and simulink: a web-based approach," 1998. [Online]. Available: http://ctms.engin.umich.edu/CTMS

[46] R. Schneider *et al.*, "Constraint-driven synthesis and tool-support for Flexray-based automotive control systems," in *(CODES+ISSS)*, 2011.

[47] K. Osman, M. F. Rahmat, and M. A. Ahmad, "Modelling and controller design for a cruise control system," *5th International Colloquium on Signal Processing & its Applications (CSPA)*, 2009.

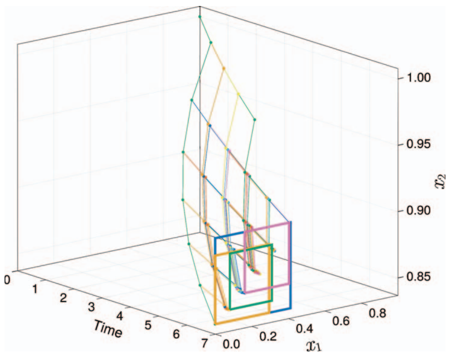


Fig. 6. An illustration of one iteration of the bounded runs algorithm.

Appendix

This appendix contains ancillary materials omitted from the main text of the paper. Figure 6 shows an example of the reachability algorithm in Section III-B. Tables III and IV give the safe deviation bounds for Sections V-A and V-B, respectively. We give descriptions of the scheduling algorithms used in Appendices A and B. Additionally, we list the plant models for our experiments in Appendix C.

A. *Job-Class-Level Scheduling*

Under job-class-level scheduling, each task τ_i with $O_i = \binom{m_i}{k_i}$ is defined as having job classes ranging from 0 to m_i if $m_i < k_i$, or only one class if $m_i = k_i$ (i.e., if the task is hard real-time). Assuming the task is not hard real-time, each job-class j contains those jobs of τ_i whose most recent sequence of consecutive deadline hits has length j. This definition allows a number of deadlines to be missed since the most recent sequence of deadline hits. To place a bound on this, a *miss threshold* w_i is used, defined as

$$w_i = \max\left\{\left\lfloor\frac{k_i}{m_i}\right\rfloor - 1, 1\right\}. \tag{6}$$

If w_i consecutive deadlines are missed, the next job is assigned to job-class 0. Figure 7 shows a schedule under this scheme, including the job-class indices and priority assigned to each job. Task 1 has parameters $(10, 10, 6, \binom{2}{4})$, and Task 2 has parameters $(7, 7, 4, \binom{3}{7})$.

The priorities of the job-classes of each task are monotonic non-increasing, so a task's priority can only decrease as it meets more consecutive deadlines. Intuitively, this increases the likelihood that a task will meet its deadline if it has not met as many recently, helping to satisfy the weakly hard constraint. Priorities for this scheduler are assigned through three heuristics. First, if the tasks are schedulable under deadline monotonic priority (where shorter deadlines get higher priority), that assignment is used. If not, then tasks are sorted in order of ascending miss threshold w_i, with ties broken by deadline. Then, for each job-class-level, priorities are assigned in descending order by task. A schedulability test, discussed below, is then run, and if the system is unschedulable, a third heuristic is tried. This heuristic holds the priority of job-classes for each task τ_i in groups of $\lceil m_i/(k_i - m_i)\rceil$ classes, allowing more jobs to run at higher priority.

As this scheduler operates in an online fashion, a schedulability test is required to verify that no runtime violations of the weakly hard constraints may occur. A sufficient schedulability test is provided in [16]. The procedure follows two stages. First, a worst-case response time is computed for each job-class of each task, following a similar strategy to time-demand analysis for task-level fixed priority systems [42]. From these worst-case response times, it is known which job-classes may miss their deadlines. The schedulability test thus concludes by checking all possible patterns of job-classes for each task τ_i, and verifying that none violates the weakly hard constraint O_i. This being the case, the task system is deemed schedulable.

B. *Automata-Based Scheduler*

As described in Section III-B, a weakly hard constraint $O = \binom{m}{k}$ can be modeled as a finite-state automaton. An accepting

TABLE III
Safe deviation bounds for RC Network, F1tenth Steering, DC Motor, Car Suspension, and Cruise Control models in Section V-A

	m				
k	1	2	3	4	5
2	0.319				
3	0.577	✓			
4	0.783	✓	✓		
5	0.945	✓	✓	✓	
6	1.070	✓	✓	✓	✓

	m				
k	1	2	3	4	5
2	1.786				
3	3.641	✓			
4	5.566	✓	✓		
5	✗	✗	✗	✗	
6	✗	✗	✗	✗	✗

	m				
k	1	2	3	4	5
2	0.005				
3	0.011	✓			
4	0.016	✓	✓		
5	0.020	✓	✓	✓	
6	0.025	0.020	✓	✓	✓

	m				
k	1	2	3	4	5
2	0.141				
3	0.334	0.111			
4	0.628	0.245	0.103		
5	0.937	0.393	0.228	0.099	
6	1.593	0.557	0.368	0.220	0.096

	m				
k	1	2	3	4	5
2	1.563				
3	3.489	1.184			
4	✗	2.776	0.983		
5	✗	4.854	2.416	0.873	
6	✗	✗	4.139	2.091	0.783

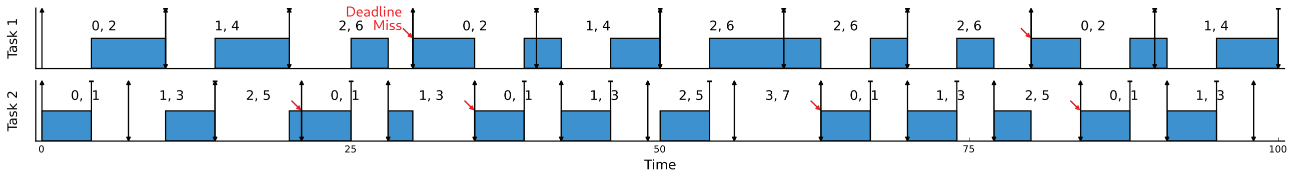


Fig. 7. A job-class-level schedule of two tasks. Each job is labeled with (job-class, priority).

run of this automaton corresponds to a sequence of deadline hits and misses satisfying O, i.e., a string in $\mathcal{L}(O)$. This scheduling algorithm begins by creating such an automaton for the weakly hard constraint $O_i = \binom{m_i}{k_i}$ of each task τ_i. Each location in each of these automata corresponds to a window of k_i deadline hits and misses in $\mathcal{L}(O_i)$. Such a window can be represented as a binary string, with 0 representing a miss and 1 representing a hit. A location corresponding to the string $\alpha\beta$, where α is a single bit and β is a string of $k_i - 1$ bits, has at most two outgoing transitions: one to $\beta 1$ on a deadline hit, and if it is in $\mathcal{L}(O_i)$, one to $\beta 0$ on a deadline miss.

Given such an automaton for each task in τ, the algorithm proceeds by taking a product of the automata. This product automaton has *vectors* of $|\tau|$ bits as its input characters, and the states correspond to vectors of $|\tau|$ strings. Each input vector has at most j bits as 1 to ensure that at most j jobs can be run in each slot. Similarly to the automata for individual controllers, each transition moves to a state whose most recent bit in each string equals the corresponding bit in the input vector. A cycle in such an automaton corresponds to an infinite-length schedule of the tasks in τ that satisfies all their weakly hard constraints. By construction, the automaton will always have such a cycle if the task system is schedulable, and thus, the scheduler is optimal under its assumptions.

C. Experimental Setup

The parameters of our control systems used in Section V in continuous time are listed in this section. Our case studies use discretized versions of these models with a one sampling period delay, according to the LET paradigm. Controllers were computed from these delayed, discretized systems using LQR with identity coefficient matrices.

1) RC Network model: Our first model is of voltages in a resistor-capacitor network [43], given in continuous time by

$$\dot{x}(t) = \begin{bmatrix} -6 & 1 \\ 0.2 & -0.7 \end{bmatrix} x(t) + \begin{bmatrix} 5 \\ 0.5 \end{bmatrix} u(t).$$

2) F1tenth Steering model: The linearized F1tenth [44] car steering model seeks to keep a car driving in a straight line, and is given in continuous time as

$$\dot{x}(t) = \begin{bmatrix} 0 & 6.5 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 19.685 \end{bmatrix} u(t).$$

3) DC Motor model: Our next model is of the angle and angular velocity of a DC motor [45], and is given by

$$\dot{x}(t) = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u(t).$$

4) Car Suspension model: To provide a higher-dimensional model, we consider a car suspension system [46], given by

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -8 & -4 & 8 & 4 \\ 0 & 0 & 0 & 1 \\ 80 & 40 & -160 & 60 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 80 \\ 20 \\ -1120 \end{bmatrix} u(t).$$

5) Cruise Control model: Finally, we consider a model of a car cruise control system [47], given by

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.0476 & -5.2856 & -0.238 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 2.4767 \end{bmatrix} u(t).$$

TABLE IV
SAFE DEVIATION BOUNDS FOR RC NETWORK, F1TENTH STEERING, DC MOTOR, CAR SUSPENSION, AND CRUISE CONTROL MODELS IN SECTION V-B

	m				
k	1	2	3	4	5
2	0.319				
3	0.577	✓			
4	0.783	✓	✓		
5	0.945	✓	✓	✓	
6	1.070	✓	✓	✓	✓

	m				
k	1	2	3	4	5
2	1.786				
3	3.641	✓			
4	5.566	✓	✓		
5	✗	✗	✗	✗	
6	✗	✗	✗	✗	✗

	m				
k	1	2	3	4	5
2	0.003				
3	0.006	0.003			
4	0.008	0.006	0.003		
5	0.011	0.008	0.006	0.003	
6	0.014	0.011	0.008	0.006	0.003

	m				
k	1	2	3	4	5
2	0.062				
3	0.127	✓			
4	0.193	0.109	✓		
5	0.274	0.178	0.097	✓	
6	0.373	0.272	✓	✓	✓

	m				
k	1	2	3	4	5
2	✗				
3	✗	4.967			
4	✗	✗	4.026		
5	✗	✗	✗	3.386	
6	✗	✗	✗	✗	3.050