Analyzing Real-Time Scheduling of Cyber-Physical Resilience

Björn Andersson
Carnegie Mellon University

Dionisio de Niz
Carnegie Mellon University

Sagar Chaki
Mentor Graphics

***Abstract*—Cyber-Physical Systems (CPS) involve software executing on a computer that interacts with its physical environment. Common steps in the design and analysis of such systems are: model the physical environment, develop software to interact with this physical environment, specify timing requirements of software, configure the software (e.g., assign priorities), and then analyze the timing requirements for a given configuration. This approach works but tends to have low resilience to disruption. With the pervasive use of CPS, there is an increasing need to develop timing analysis methods that achieve increased resilience by modeling the linkage between the execution of software and the physical environment. In this paper, we present a new model that describes the current state of the physical environment in terms of how tolerant it is to disruption of the software system; we call this model *Cyber-Physical Resilience* (CPR). We present an exact schedulability test for this model and implement a tool that performs this schedulability test. We evaluate it on randomly-generated tasksets and on a model of a multi-UAV system from [1].**

I. Introduction

Cyber-Physical Systems (CPS) involve software executing on a computer that interacts with its physical environment. Common steps [15] in the design and analysis of such systems are: model the physical environment, develop software to interact with this physical environment, specify timing requirements of software, configure the software (e.g., assign priorities), and then analyze the timing requirements for a given configuration. This approach works but tends to have low resilience to disruption. With the pervasive use of CPS, there is an increasing need to develop timing analysis methods that achieve increased resilience by modeling the linkage between the execution of software and the physical environment.

The research community has (i) explored ideas on how scheduling and task set parameters impacts performance metrics of the physical world (e.g., control performance) [13], [10], (ii) developed new ways of scheduling that is suited for the performance of the physical world, [8], [7], [4], [3], [16], [2], [11], [17], (iii) developed software architectures and runtime systems that allow unverified controllers to be used in safety-critical systems [6], [14], and (iv) developed scheduling with skips [5], [9]. But there has not been much focus on resilience. From the perspective of resilience, the ideas in [12] are particularly interesting—for this reason, we describe then now.

Figure 1 shows notions in [12]. Consider a plant. Let ϕ be a correctness property of the plant (e.g., a UAV stays within a safe geographic region). We represent ϕ as a set of states

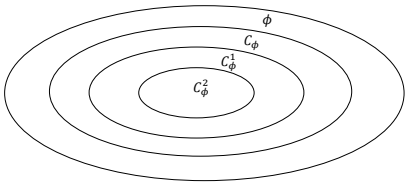

Fig. 1. Sets of states of the physical plant/environment.

of the plant where this correctness property is true. There is also a computer system that periodically senses and performs actions on the plant. For a given pair of state and action there is a set of possible successor states.

Note that the state refers to the state of the plant—not the state of the computer system. Let C_ϕ be a subset of ϕ such that for each state in C_ϕ there is an action such that the set of possible successor states is a subset of C_ϕ.

Let a *null* action be an action such that the software does nothing. The interpretation of a null action is application specific and in particular, it depends on the actuator. For example, an actuator may be designed so that if it receives no new command, then it re-applies the command produced by the software in the previous period. Alternatively, an actuator may have an internal timeout so that if it has not received a new command from the software within a given timeout, then it actuates a specific command. For example, if a motor has not received any new command within a given timeout, then it may disengage. Let C_ϕ^k (for $k \geq 1$) denote the set of states such that for each state in C_ϕ^k, it holds that if k null actions are taken, then each possible successor state is in C_ϕ. Thus, both C_ϕ^2 and C_ϕ^1 are sets of safe states but C_ϕ^2 is more resilient in the sense that it can tolerate more null actions without the plant reaching an unsafe state.

Unfortunately, Lucia et al. [12] studied a system with a single physical environment and a single task. We believe it is desirable to have a model describing multitasking in a way that extends the ideas in [12] and also create a corresponding schedulability test.

Therefore, in this paper, we present a new task model inspired by the ideas in [12] but we focus on real-time tasks, enforcement, interaction with physical environment, and schedulability analysis. Our model describes the relationship between the timing behavior and the physical resilience of the system; hence we name it the *Cyber-Physical Resilience*

(CPR) model. For this model, we present an exact schedulability analysis. The main idea of our schedulability test is as follows: (i) identify necessary conditions for a failure, (ii) formulate a Satisfiability Modulo Theories (SMT) instance such that if the necessary condition of failure is true then the SMT instance is satisfiable, (iii) by taking the contrapositive of the previous condition, obtain that if the SMT instance is unsatisfiable then the taskset is schedulable (this is our new schedulability test), and (iv) prove that our new schedulability test is exact. We also implement a tool that performs this schedulability test. We evaluate it on randomly-generated tasksets and on a model of a multi-UAV system from [1].

We consider this research to be significant because it allows the real-time systems community to develop new results on real-time scheduling that considers the interaction between the software and its physical environment—yet do so with an application-independent abstraction rather than specific types of differential equations of the physical world.

The remainder of this paper is structured as follows. Section II gives a background on the model in [12] and our observations. Section III formally presents our system model. Section IV presents our new schedulability analysis. Section V presents a tool that performs the schedulability analysis and presents evaluation. Section VI concludes the paper.

II. Physical Interaction

Lucia et al. [12] studied control of such a system where the computer can suffer from a Denial-of-Service attack, and it is desired to ensure that the system is still in a safe state after that. Recall that Figure 1 shows notions that they used.

We will now describe how to use this idea to create a model for the real-time scheduling of the resilience of this type of systems. We will describe a system where the software consists of a set of tasks where each task generates a sequence of jobs. We assume that tasks operate on different plants, that is, task τ_i is associated with a plant i with the correctness property ϕ_i. A counter is associated with each task; if the counter associated with task τ_i is equal to k, then it means that plant i is in a state in $C^k_{\phi_i}$. A job can be skipped and then it outputs a null action. If a job is skipped, the counter of the corresponding task is decremented. If a job is not skipped it can happen that the job finishes with some margin before its deadline issuing a normal controller action; then the counter is incremented. If the job is not skipped and the job does not finish with some margin before its deadline, then within some margin before the deadline, the job is killed and an enforcer arrives; if the enforcer finishes before the deadline issuing an enforcer action, then the counter is unchanged. If the enforcer has not yet finished at its deadline, then the enforcer is killed (and a null action is output) and the counter is decremented.

We assume that each task is assigned a priority and fixed-priority preemptive scheduling is used. We assume that whether a job is skipped cannot be controlled by the scheduler but we have rules that bound this behavior.

We are interested in having very few assumptions on knowledge of execution times while still being able to provide pre-run-time guarantees that at each instant, for each task, the counter of the task at this instant is non-negative. For this purpose, we can stipulate that if a job is of the type respectC (intuitively respect execution time), then the execution time is at most a given parameter; if the job is of the type not-respectC then execution time may be greater than this parameter. Clearly, if all jobs are not-respectC, then it is impossible to provide pre-run-time guarantees. Therefore, we also stipulate bounds on how many jobs can be not-respectC.

Typically, it is desired not only to maintain the plant in a safe state but also to achieve other objectives (for example, a UAV should stay within a geographical area but it should also follow waypoints). Hence, there are cases (where the plant is not close to the border of the set of safe states), when a successful action (a job finishing with some margin before its deadline) should not increment the counter of the task. We can model this by introducing a parameter per task and when the counter of a task has reached this parameter, then the counter is not permitted to be incremented further.

Our goal is to (i) formulate a task model based on the above ideas, (ii) present an exact schedulability test for this task model, and (iii) evaluate the new schedulability test.

III. System Model

Subsection III-A states notations that we will use. Subsection III-B states taskset parameters. Subsection III-C presents run-time behavior; it explains the meaning of taskset parameters. Subsection III-D defines the notion *schedulable*.

A. Notation

Throughout this article, we use the following notation and abbreviations. "with respect to" is written as wrt. "left-hand side" is written as lhs. "right-hand side" is written as rhs. $\langle a, b\rangle$ is a tuple with two elements a and b. $[a, b]$ is an interval of real numbers. $\{a..b\}$ is the set of integers $\geq a$ and $\leq b$.

We use dot to mean it holds that; for example, when we write $\forall x\ Q(x).\ P(x)$ we mean forall x such that $Q(x)$ is true, it holds that $P(x)$ is true. In some cases, when Q is a set, we write $\forall x \in Q.\ P(x)$ we mean forall x such that x is in the set Q, it holds that $P(x)$ is true. We will frequently use the above notations on tuples, for example when we write $\forall\langle j, q\rangle$ $Q(j, q).\ P(j, q)$ we mean forall tuples $\langle j, q\rangle$ such that $Q(j, q)$ is true, it holds that $P(j, q)$ is true. We will assume that logical conjunction can be performed over a set of variables; if the set is empty, then the result is true. Ditto for logical disjunction.

In figures that show schedules, an arrow pointing upwards indicates the arrival of a job, an arrow pointing downwards indicates the absolute deadline of a job, and a solid vertical line will be used to indicate a time when the run-time system releases an enforcement execution (if needed) for a job. When we say increment without specifying the amount, it is assumed to mean increment by one. Ditto for decrement.

B. Static parameters

Table I shows an example of a system in our model. We consider a taskset τ and a single processor. Each task $\tau_i \in \tau$

$|\tau| = 2$
$\text{prio}_1 = 2\ T_1 = 1.0\ D_1 = 0.8\ Z_1 = 0.64\ C_1 = 0.50\ E_1 = 0.10\ \text{MAXCOUNT}_1 = 4\ \text{RC}_1 = 1$
$\text{prio}_2 = 1\ T_2 = 2.2\ D_2 = 1.7\ Z_2 = 1.40\ C_2 = 0.61\ E_2 = 0.25\ \text{MAXCOUNT}_2 = 1\ \text{RC}_2 = 1$

TABLE I
AN EXAMPLE OF A SYSTEM IN OUR MODEL.

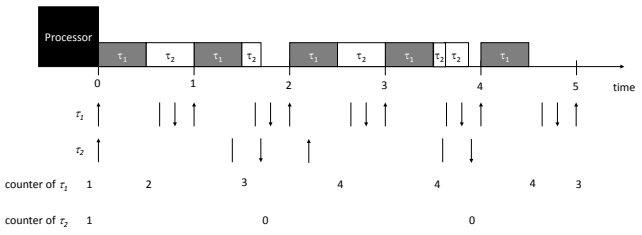

Fig. 2. Illustration of a schedule that the system can generate. τ_1 has the highest priority; hence, a job of τ_1 executes immediately when it arrives. It can be seen that for each of the first five jobs of τ_1, the finishing time is at most Z_1 after its arrival. Because of this early finishing, for each of the first three jobs of τ_1, the counter of τ_1 is incremented. For the 4^{th} and 5^{th} job of τ_1, it holds that the job also has early finishing time but since the counter has already become 4 and $\text{MAXCOUNT}_1 = 4$, it follows that the counter is not incremented further. The 6^{th} job of τ_1 arrives at time 5; this job is skipped and hence the counter of τ_1 is decremented. As a result, the counter of τ_1 becomes 3 at time 5. Note that in this schedule, it never happens that a job of τ_1 is eligible Z_1 time units after its arrival; hence, jobs of τ_1 never perform enforcement execution. τ_2 has lower priority; hence a job of τ_2 can only execute when a job of τ_1 is not executing. For the 1^{st} job of τ_2, it holds that it has not yet finished Z_2 time units after its arrival; hence this job is killed and its enforcement execution is released (at time 1.4). Note that this job cannot execute immediately after time 1.4 because of the higher priority task. Therefore, this job has to wait until time 1.5 and then it performs enforcement execution during [1.5,1.7]. At time 1.7, the deadline of the 1^{st} job of τ_2 expires and hence this job is killed and as a result, the counter of τ_2 is decremented. The 2^{nd} job of τ_2 has similar behavior but since its enforcement execution experiences less interference, it finishes before the deadline and hence the counter of τ_2 is not changed.

is characterized by prio_i, T_i, D_i, Z_i, C_i, E_i, MAXCOUNT_i, and RC_i such that $(T_i \geq D_i > Z_i \geq C_i \geq 0) \wedge (Z_i > 0) \wedge (D_i - Z_i \geq E_i \geq 0) \wedge (\text{MAXCOUNT}_i \in \mathbb{N}_{\geq 1}) \wedge (\text{RC}_i \in \mathbb{N}_{\geq 1})$. The interpretation is as follows. A task τ_i is assigned priority prio_i. A task τ_i generates a sequence of jobs where two consecutive jobs of τ_i have arrival times separated by at least T_i time units and each job of τ_i has relative deadline D_i. If a job of τ_i is a respectC job, then C_i is an upper bound on the execution time of this job performed in the time interval from its arrival until Z_i time units after its arrival. For a job of τ_i, regardless of whether it is a respectC job, it holds that E_i is an upper bound on the execution time of this job performed in the time interval from Z_i time units after its arrival until D_i time units after its arrival. MAXCOUNT_i is the largest value that the counter of task τ_i may take at run-time. RC_i specifies that we assume it cannot happen that RC_i consecutive jobs of τ_i are not-respectC. We assume that priorities of tasks are unique, that is, $((i \neq j) \wedge (\tau_i \in \tau) \wedge (\tau_j \in \tau)) \Rightarrow (\text{prio}_i \neq \text{prio}_j)$. For convenience, we let $\text{hp}(i)$ denote the set of indices of tasks with higher priority than task τ_i and $\text{hep}(i) = \text{hp}(i) \cup \{i\}$.

C. Run-time behavior

Figure 2 shows a schedule that the taskset specified by Table I can generate. $\tau_{i,q}$ denotes the q^{th} job of task τ_i. The priority of a $\tau_{i,q}$ is equal to prio_i. A job can be a skipped job or not; if it is skipped, then there is a time at which it gets skipped. (For example, a job may be skipped when it arrives because the operating system decided this. Alternatively, the application software may decide that a job should be skipped and this type of skip will happen after the job has arrived.) A job can be respectC job or not. We say that a job of task τ_i *Zexpires* at time Z_i plus the arrival time of the job. Analogously, we also say that a job of task τ_i *Dexpires* at time D_i plus the arrival time of the job. For a job $\tau_{i,q}$, the normal mode of the job is the time interval from its arrival until it Zexpires. For a job $\tau_{i,q}$, the enforcement mode of the job is the time interval from when it Zexpires until it Dexpires. A job performs normal execution in its normal mode. A job performs enforcement execution in its enforcement mode. The following rules (for evolution of counters, whether a job is eligible, and how a job is selected for execution) apply at run-time:

1) When the system starts, the counter of τ_i is initialized to some non-negative value. The choice in this value may be done non-deterministically when the system starts.
2) If a job $\tau_{i,q}$ is skipped, then it must have been that the counter of τ_i was positive just before that.
3) When a job arrives, if it is not skipped when it arrives, it becomes eligible.
4) When a job arrives, if it is skipped when it arrives, it is set to non-eligible.
5) If $\text{RC}_\text{i} = 1$, then each job of τ_i is a respectC job.
6) If $\text{RC}_\text{i} \geq 2$ and at least $\text{RC}_\text{i} - 1$ jobs of τ_i have arrived and $\tau_{i,q}$ is a new job arriving, and all $\text{RC}_\text{i} - 1$ most-recent preceding jobs of τ_i are not-respectC job, then $\tau_{i,q}$ is a respectC job.
7) If for $\tau_{i,q}$, the job is in its normal mode and the job is skipped, then it finishes, it becomes non-eligible, and the counter of τ_i is decremented. (A job can only be skipped in its normal mode.)
8) If for $\tau_{i,q}$, the job is in its normal mode and it is not skipped and the job has performed C_i units of normal execution, then the job finishes (note that a job can finish even if it performs less) and it becomes not-eligible.
9) If for $\tau_{i,q}$, the job finishes in its normal mode and it is not skipped and the counter of the τ_i is at most $\text{MAXCOUNT}_i - 1$, then the counter of τ_i is incremented. (Note that if the counter of τ_i equals MAXCOUNT_i, then the counter is unchanged; hence, the counter cannot exceed MAXCOUNT_i.)
10) When a job Zexpires, it leaves its normal mode and

enters its enforcement mode.

11) If for $\tau_{i,q}$, the job is in its enforcement mode and the job has performed E_i units of enforcement execution, then the job finishes (note that a job can finish even if it performs less; also note that a job cannot be skipped in its enforcement mode) and it becomes not-eligible.
12) For $\tau_{i,q}$, if the job finishes in its enforcement mode, then the counter of τ_i does not change.
13) For $\tau_{i,q}$, when the job Dexpires, it finishes (here we say it is killed) and it becomes not-eligible and the counter of τ_i is decremented.
14) A job executes if and only if the job is eligible and there is no higher-priority job that is eligible at that time.

D. Schedulable

Informally, a taskset is schedulable if for all possible schedules that the taskset can generate, it holds that at each instant, for each task, the counter of the task at this instant is non-negative. Thus, we start by defining terminology for defining possible schedules.

Let R be an assignment, for each task, the number of jobs it generates and for each job, its arrival time and execution time in normal mode and in enforcement, whether it is a skip-job and the time when it becomes a skip job (only used if it is a skip job) and respectC. $\mathrm{nj}_i(R)$ denotes the number of jobs generated by τ_i for assignment R. Let $A_{i,q}(R)$ be the arrival time of $\tau_{i,q}$ for assignment R. Let $c_{i,q}(R)$ be the execution requirement of $\tau_{i,q}$ in normal mode for assignment R. Let $e_{i,q}(R)$ be the execution requirement of $\tau_{i,q}$ in enforcement mode for assignment R. Let $\mathrm{skip}_{i,q}(R)$ be a Boolean indicating whether $\tau_{i,q}$ is a skip-job for assignment R. Let $\mathrm{skipt}_{i,q}(R)$ be a real number indicating the time when $\tau_{i,q}$ becomes a skip-job (only used if it is a skip job) for assignment R. Let $\mathrm{respC}_{i,q}(R)$ be a Boolean indicating whether $\tau_{i,q}$ is respectC for assignment R. Let $\mathrm{leg}(R,\tau)$ mean legal assignment. Formally:

$$\begin{aligned}\mathrm{leg}(R,\tau) = {} & \big(\forall\langle i,q\rangle\ (\tau_i \in \tau) \wedge (q \in \{2..\mathrm{nj}_i(R)\})\ A_{i,q}(R) - A_{i,q-1}(R) \geq T_i\big) \wedge \\ & \big(\forall\langle i,q\rangle\ (\tau_i \in \tau) \wedge (q \in \{1..\mathrm{nj}_i(R)\})\ (c_{i,q}(R) \geq 0) \wedge (\mathrm{respC}_{i,q}(R) \Rightarrow (c_{i,q}(R) \leq C_i))\big) \wedge \\ & \big(\forall\langle i,q\rangle\ (\tau_i \in \tau) \wedge (q \in \{1..\mathrm{nj}_i(R)\})\ e_{i,q}(R) \in [0, E_i]\big) \wedge \\ & \big(\forall\langle i,q\rangle\ (\tau_i \in \tau) \wedge (q \in \{1..\mathrm{nj}_i(R)\})\ \mathrm{skipt}_{i,q}(R) \in [A_{i,q}(R), A_{i,q}(R) + Z_i]\big) \wedge \\ & \big(\forall\langle i,q\rangle\ (\tau_i \in \tau) \wedge (q \in \{1..\mathrm{nj}_i(R) - (\mathrm{RC}_i - 1)\})\ \vee_{q' \in \{q..q+\mathrm{RC}_i-1\}}\ \mathrm{respC}_{i,q'}(R)\big)\end{aligned}$$

Let ca be a counter assignment; that is, it is an assignment for each task, for each time, the value of the counter of the task at that time. Let $\mathrm{legca}(\mathrm{ca}, R, \mathrm{sc}, \tau)$ mean that ca is legal counter assignment related to the assignment R, schedule sc, τ; it means that the counter assignment respects the increment and decrement rules specifies in the Subsection III-C.

If R is an assignment, ca is a counter assignment, and sc is a schedule, then when we say that $\langle R, \mathrm{ca}, \mathrm{sc}\rangle$ is legal wrt τ, we mean that all three conditions below are true:

1) R is legal wrt to τ.
2) ca is legal wrt R, sc, and τ.
3) sc can be generated from R.

We say that τ is *schedulable* if and only if for each lega l$\langle R, \mathrm{ca}, \mathrm{sc}\rangle$, it holds that for each task $\tau_i \in \tau$, at all times, the counter of τ_i is non-negative.

A schedulability test is a function that takes τ as input and outputs a Boolean. For an *exact* schedulability test, it holds that if and only if the schedulability test outputs true, then the system is schedulable.

IV. New Schedulability Test

In this section, we present our new schedulability test for the CPR model. Our plan is as follows. If a taskset is unschedulable, then there is a schedule in which there is a failure (i.e., a counter becomes negative). We reason (in Section IV-A) about this failure and obtain the existence of another schedule that the system can generate for which there is also a failure. We represent (in Section IV-B) those schedules with variables and constraints. Thus, if a taskset is unschedulable, then this constraint satisfaction problem is satisfiable. Then (in Section IV-C) we show that we have obtained a necessary condition for an unschedulable taskset. We take the contrapositive of this necessary condition and this yields a sufficient condition for a taskset to be schedulable. We then show that this condition is also exact. We also present an algorithm to evaluate this condition; this yields our exact schedulability test for the CPR model.

A. Reasoning about Failure

Consider an unschedulable taskset τ. From the definition of schedulability, it follows that

> There is an assignment R, a counter assignment ca, a schedule sc, a task τ_{iD}, and an instant t_0 such that (i) R is legal wrt the taskset τ, (ii) ca is legal wrt R, sc, and τ, (iii) sc can be generated by R and τ, and (iv) just before t_0, the counter of τ_{iD} was 0 and at t_0, the counter of τ_{iD} becomes -1.

If there are multiple such t_0, then choose the earliest one. Hence, for each task, for each instant before t_0, it holds that at this instant the counter of this task is non-negative. We will present transformations such that after each transformation, the above conditions are true but in addition, there are other conditions that are true as well. We will now perform a transformation as specified by the following steps:

- T1 For tasks of lower priority than τ_{iD}, set the number of jobs to zero.
- T2 Remove all jobs that arrive after t_0.
- T3 For each task, set the counter of this task as a function of time to reflect the changes of steps T1 and T2.

Note that even after this change, it holds that at time t_0, the counter of τ_{iD} becomes -1. Recall that according to our system model, there are two reasons why the counter of a task is decremented: either because a job of this task is skipped or because the job finishes too late. The former cannot happen when the counter is zero (which is the case just before time t_0). Hence, at time t_0, the counter of τ_{iD} is decremented because a job of τ_{iD} finishes too late. We let qD denote the index of this job of task τ_{iD}. In addition, recall from our system model that if a job has not finished by its absolute deadline, then the job is killed at its absolute deadline; thus $\tau_{\mathrm{iD,qD}}$ is killed a time t_0. We will now perform a transformation as specified by the following steps:

T4 For all jobs except $\tau_{\mathrm{iD,qD}}$, if the job performs some enforcement execution at time t_0 or later, then reduce the enforcement execution time of the job; keep doing this until it performs no enforcement execution after t_0.

T5 For all jobs except $\tau_{\mathrm{iD,qD}}$, if the job performs some normal execution at time t_0 or later, then reduce the normal execution time of the job; keep doing this until it performs no normal execution after t_0.

T6 For each task, set the counter of this task as a function of time to reflect the changes of steps T4-T5.

Note that even after this change, it holds that at time t_0, the counter of τ_{iD} becomes -1. Hence, it holds that:

> There is an assignment R, a counter assignment ca, a schedule sc, a task τ_{iD}, an instant t_0, and a positive integer qD such that (i) R is legal wrt the taskset τ, (ii) ca is legal wrt R, sc, and τ, (iii) sc can be generated by R and τ, (iv) just before t_0, the counter of τ_{iD} was 0 and at t_0, the counter of τ_{iD} becomes -1, (v) for each task, for each instant before t_0, it holds that at that time, the counter of the task is non-negative, (vi) for each task of lower priority than τ_{iD} it holds that, the task generates no jobs, (vii) no jobs arrive after t_0, (viii) the number of jobs generated by τ_{iD} is qD, (ix) $\tau_{\mathrm{iD,qD}}$ is not a skipjob, (x) the absolute deadline of $\tau_{\mathrm{iD,qD}}$ is t_0, (xi) the amount of enforcement execution performed by $\tau_{\mathrm{iD,qD}}$ upto time t_0 is strictly less than $e_{\mathrm{iD,qD}}(R)$, (xii) $\tau_{\mathrm{iD,qD}}$ is killed at time t_0, and (xiii) the processor is idle at all times strictly after t_0.

Let us consider two cases based on the initial value of the counter of task τ_{iD}.

C1: When the system starts, the value of the counter of task τ_{iD} is at least 1.
Since at time t_0, it holds that the counter of τ_{iD} changes from 0 to -1 and since the counter of τ_{iD} started with at least 1, it holds that there are at least two jobs of τ_{iD} for which the counter of τ_{iD} was decreased. Hence $\mathrm{qD} \geq 2$. Let us define qD as the smallest number such that after the absolute deadline of $\tau_{\mathrm{iD,qD'}}$ until t_0, the counter of task τ_{iD} does not change. Formally,

$$\mathrm{qD}' = \min_{q \mid \textit{the counter of } \tau_{\mathrm{iD}} \textit{ does not change during } (A_{\mathrm{iD},q}(R)+D_{\mathrm{iD}},t_0)} q$$

Note that since there are at least two jobs of τ_{iD} for which the counter of τ_{iD} was decreased, it follows that qD' exists. We will now perform a transformation as specified by the following steps:

T7 Set $\tau_{\mathrm{iD,qD'}}$ to be skipped; set the time when this job is skipped to be the time when the job arrives.

T8 Set $\tau_{\mathrm{iD,qD'}}$ to respectC.

T9 Remove all jobs $\tau_{\mathrm{iD},q}$ such that $((q \neq \mathrm{qD}') \wedge (q \neq \mathrm{qD}))$.

T10 Set the counter of τ_{iD} as a function of time to reflect the changes of steps T7-T9.

About the transformation, we note the following:

A The job $\tau_{\mathrm{iD},1}$ after this transformation refers to the same job as $\tau_{\mathrm{iD,qD'}}$ before the transformation.

B The job $\tau_{\mathrm{iD},2}$ after this transformation refers to the same job as $\tau_{\mathrm{iD,qD}}$ before the transformation.

C After the transformation, at time t_0, the counter of τ_{iD} changes from 0 to -1.

D After the transformation, there are two jobs of τ_{iD}.

E After the transformation, each of the two jobs of τ_{iD} causes the counter of τ_{iD} to be decremented.

F After the transformation, when the system starts, the counter of τ_{iD} is 1 (this follows from C. and E.).

G After the transformation, $\tau_{\mathrm{iD},1}$ does not execute (this follows from A and T7).

H After the transformation, $\tau_{\mathrm{iD},1}$ is a respectC job. (this follows from A and T8).

I If $\mathrm{RC_{iD}} = 1$, then before the transformation, each job of τ_{iD} is a respectC job. (follows from the fact that R is legal).

J If $\mathrm{RC_{iD}} = 1$, then after the transformation, $\tau_{\mathrm{iD},2}$ is a respectC job. (follows from I and B).

We will now show that after this transformation, the assignment and schedule are still legal. Clearly, this transformation does not change the arrival times, execution times of tasks with index in $\mathrm{hp(iD)}$; also, for these tasks, the times when they execute do not change, their counters do not change, and their respectC do not change (and hence RC constraints are satisfied too). We will now show that this also holds for the task τ_{iD}. Clearly, the execution time and arrival time constraints for τ_{iD} are satisfied. We will now show that the RC constraint for τ_{iD} is satisfied after the change. Consider the case that $\mathrm{RC_{iD}} = 1$. From H. and J., it follows that after the transformation, both jobs of τ_{iD} are respectC jobs; hence the RC constraint for τ_{iD} is satisfied. Consider the case that $\mathrm{RC_{iD}} \geq 2$. In order to prove that the RC constraint of τ_{iD} is satisfied after the change, it suffices to show that for each sequence of $\mathrm{RC_{iD}}$ jobs of τ_{iD}, it holds that at least one of these jobs is a respectC job. Note (from H) that $\tau_{\mathrm{iD},1}$ after the transformation is a respectC job and hence each such sequence has a respectC job. Thus, the RC constraint of τ_{iD} is satisfied after the transformation. Hence, it holds that:

> There is an assignment R, a counter assignment ca, a schedule sc, a task τ_{iD}, and an instant t_0 such that (i) R is legal wrt the taskset τ, (ii) ca is legal wrt R, sc, and τ, (iii) sc can

be generated by R and τ, (iv) just before t_0, the counter of τ_{iD} was 0 and at t_0, the counter of τ_{iD} becomes -1, (v) for each task, for each instant before t_0, it holds that at that time, the counter of the task is non-negative, (vi) for each task of lower priority than τ_{iD} it holds that, the task generates no jobs, (vii) no jobs arrive after t_0, (viii) the number of jobs generated by τ_{iD} is 2, (ix) $\tau_{\mathrm{iD},2}$ is not a skipjob, (x) the absolute deadline of $\tau_{\mathrm{iD},2}$ is t_0, (xi) the amount of enforcement execution performed by $\tau_{\mathrm{iD},2}$ upto time t_0 is strictly less than $e_{\mathrm{iD},2}(R)$, (xii) $\tau_{\mathrm{iD},2}$ is killed at time t_0, (xiii) the processor is idle at all times strictly after t_0, and (xiv) when the system starts, the counter of τ_{iD} is 1.

Let t_{-1} denote the earliest time instant such that at each instant during [t_{-1},t_0], the processor is busy. We will now discuss the arrival time of $\tau_{\mathrm{iD},1}$ and $\tau_{\mathrm{iD},2}$. Let us define Δ as $A_{\mathrm{iD},2}(R) - t_{-1}$. From the definition of t_{-1} and from the fact that we consider work-conserving scheduling, it follows that $\Delta \geq 0$. We will now consider two cases and through reasoning, we will show that after these cases, we end up with $A_{\mathrm{iD},2}(R) = t_{-1}$.

C1a: $\Delta = 0$

Using the knowledge of the case ($\Delta = 0$) and the definition of Δ yields that $A_{\mathrm{iD},2}(R) = t_{-1}$.

C1b: $\Delta > 0$

For this case, perform a transformation as specified by the following steps:

- T11 Decrement $A_{\mathrm{iD},1}(R)$ by Δ.
- T12 Decrement $A_{\mathrm{iD},2}(R)$ by Δ.
- T13 Set t_0 to the absolute deadline of $\tau_{\mathrm{iD},2}$.
- T14 Apply T4,T5,T6 so that there is no execution after the time given by the new value of t_0.
- T15 Set the counter of τ_{iD} as a function of time to reflect the changes of steps T11-T14.

Given that both jobs of τ_{iD} have their arrival times decreased by the same amount and the minimum inter-arrival time was respected before the transformation, the minimum inter-arrival time is still respected after the transformation. Also, after the transformation, we obtain $A_{\mathrm{iD},2}(R) = t_{-1}$.

Hence, regardless of the case, we end up with $A_{\mathrm{iD},2}(R) = t_{-1}$. Also, note that we end up with that the absolute deadline of $\tau_{\mathrm{iD},2}$ equals t_0 (this can be seen as follows: in Case 1a, this was true initially and we did not change it; in Case 1b, this was true initially and then we changed it—with T12—and then T13 made sure it is true). We will now perform a transformation as specified by the following steps:

- T16 Remove the job $\tau_{\mathrm{iD},1}$.
- T17 Remove all jobs arriving before t_{-1}.
- T18 Left shift-the schedule by t_{-1} time units.
- T19 For each task, set the counter of this task as a function of time to reflect the changes of steps T16-T18.

Note that after this transformation, we obtain that τ_{iD} generates a single job that arrives at time 0; at that time, the counter of τ_{iD} is zero; and D_{iD} time units later, the deadline of the single job of τ_{iD} expires at time D_{iD}. Hence, it holds that:

There is an assignment R, a counter assignment ca, a schedule sc, and a task τ_{iD} such that (i) R is legal wrt the taskset τ, (ii) ca is legal wrt R, sc, and τ, (iii) sc can be generated by R and τ, (iv) just before D_{iD}, the counter of τ_{iD} was 0 and at D_{iD}, the counter of τ_{iD} becomes -1, (v) for each task, for each instant before D_{iD}, it holds that at that time, the counter of the task is non-negative, (vi) for each task of lower priority than τ_{iD} it holds that, the task generates no jobs, (vii) no jobs arrive after D_{iD}, (viii) the number of jobs generated by τ_{iD} is 1, (ix) $\tau_{\mathrm{iD},1}$ is not a skipjob, (x) the absolute deadline of $\tau_{\mathrm{iD},1}$ is D_{iD}, (xi) the amount of enforcement execution performed by $\tau_{\mathrm{iD},1}$ upto time D_{iD} is strictly less than $e_{\mathrm{iD},1}(R)$, (xii) $\tau_{\mathrm{iD},1}$ is killed at time D_{iD}, (xiii) the processor is idle at all times strictly after D_{iD}, (xiv) when the system starts at time zero, the counter of τ_{iD} is 0, (xv) $\tau_{\mathrm{iD},1}$ arrives at time zero, and (xvi) no job arrives before time 0. [End-of-Case-1]

C2: When the system starts, the value of the counter of task τ_{iD} is 0.

If $\mathrm{qD} \geq 2$, then remove all jobs of τ_{iD} except $\tau_{\mathrm{iD,qD}}$. Now we have a situation with a single job of τ_{iD}. Let t_{-1} denote the earliest time such that the processor is busy during [t_{-1}, t_0]. Then, remove all jobs that arrive before t_{-1}. And then left-shift the schedule by t_{-1}. This yields the same situation as in the end of Case 1. [End-of-Case-2]

It can be seen that regardless of the case, we obtain that the statement just before [End-of-Case-1] is true. Let us now discuss skiptime of a job of a task in $\mathrm{hp(iD)}$—recall that if a job is skipped, then it is skipped at its skiptime; if the job is not skipped, then its skiptime neither influences the schedule nor the evolution of the counter. We will show that for each job, we can set skiptime of a job $\tau_{i,q}$ to its arrival time or finishing time. Given a job $\tau_{i,q}$, consider three cases (i) $\tau_{i,q}$ is skipped and its skiptime equals its arrival time, (ii) $\tau_{i,q}$ is skipped and its skiptime is after its arrival time, (iii) $\tau_{i,q}$ is not skipped. For case (i), we do nothing. For case (ii), we know that there can be no execution of $\tau_{i,q}$ after its skiptime. Hence, the finishing time of $\tau_{i,q}$ is at most its skiptime. We

can reduce the skiptime of $\tau_{i,q}$ until it reaches its finishing. This does not change the schedule. For case (iii), we set its skiptime to its arrival time. Since the job is not a skipjob, the value of skiptime has no impact and hence this change of skiptime does not have any impact on the schedule. It can be seen that regardless of the case, we obtain that for a job, its skiptime equals its arrival time or finishing time.

We will now discuss the counters. For each $j \in \mathrm{hp(iD)}$ do the following (i) throughout the schedule, find the smallest value of the counter for task τ_j and then (ii) throughout the schedule, subtract the counter of τ_j by the number computed in (i). Hence, we obtain that for each task with index $j \in \mathrm{hp(iD)}$, there exists a time when the counter of the τ_j is zero and the counter is never lower than 0. Also, it is easy to see that for each task with index j, it holds that there are at most $\lceil D_{\mathrm{iD}}/T_j \rceil$ jobs of task τ_j. Hence, for each task with index j, during its schedule, it holds that at each instant, the counter of task τ_j is at most $\lceil D_{\mathrm{iD}}/T_j \rceil$. Also, from the system model, for each task with index j, during its schedule, it holds that at each instant, the counter of task τ_j is at most $\mathrm{MAXCOUNT}_j$. Putting them together yields that for each task with index j, during its schedule, it holds that at each instant, the counter of task τ_j is at most $\min(\lceil D_{\mathrm{iD}}/T_j \rceil, \mathrm{MAXCOUNT}_j)$. Thus:

There is an assignment R, a counter assignment ca*, a schedule* sc*, and a task τ_{iD} such that (i) R is legal wrt the taskset τ, (ii)* ca *is legal wrt R,* sc*, and τ, (iii)* sc *can be generated by R and τ, (iv) just before D_{iD}, the counter of τ_{iD} was 0 and at D_{iD}, the counter of τ_{iD} becomes -1, (v) for each task, for each instant before D_{iD}, it holds that at that time, the counter of the task is non-negative, (vi) for each task of lower priority than τ_{iD} it holds that, the task generates no jobs, (vii) no jobs arrive after D_{iD}, (viii) the number of jobs generated by τ_{iD} is 1, (ix) $\tau_{\mathrm{iD},1}$ is not a skipjob, (x) the absolute deadline of $\tau_{\mathrm{iD},1}$ is D_{iD}, (xi) the amount of enforcement execution performed by $\tau_{\mathrm{iD},1}$ upto time D_{iD} is strictly less than $e_{\mathrm{iD},1}(R)$, (xii) $\tau_{\mathrm{iD},1}$ is killed at time D_{iD}, (xiii) the processor is idle at all times strictly after D_{iD}, (xiv) when the system starts at time zero, the counter of τ_{iD} is 0, (xv) $\tau_{\mathrm{iD},1}$ arrives at time zero, (xvi) no job arrives before time 0, (xvii) for each task with index in* $\mathrm{hp(iD)}$*, for each job of the task, the skiptime of the job is its arrival time or its finishing time, (xviii) for the job $\tau_{\mathrm{iD},1}$, its skiptime equals its arrival time, and (xix) for each task with index j, at each instant, the counter of the task at this instant is at most* $\min(\lceil D_{\mathrm{iD}}/T_j \rceil, \mathrm{MAXCOUNT}_j)$.

If we could find a simple function such that this function takes parameters of the taskset as input and outputs a Boolean such that (0) implies that the function is true, then we can obtain a schedulability test by negating the function. One could imagine that such a function could be obtained by summing up all the computation by jobs of tasks in $\mathrm{hp(iD)}$ and adding the normal and enforcement execution of $\tau_{\mathrm{iD},1}$ and then compare with D_{iD}. Unfortunately, doing so is very complicated in our model. The reason is that in our model, the amount of execution of a job depends on when the job finishes (if the job finishes before it Zexpires, then it performs no enforcement execution). Therefore, we will, instead, express the schedule in (0) with variables and constraints so that if and only if (0) is true, then the constraints are satisfiable.

B. Representing Schedules

General idea. We will express the schedule in (0) with variables and constraints so that if and only if (0) is true, then the constraints are satisfiable. We will present the constraints on a form so that satisfiability can be checked with a Satisfiability Modulo Theories (SMT) solver. Recall that (0) considers a schedule in the time interval [0,D_{iD}] and no jobs arrive before time 0 and there is no execution after time D_{iD}. Hence, we only need to consider at most $\lceil D_{\mathrm{iD}}/T_j \rceil$ jobs of task τ_j. Recall also from (0) that there is a single job of τ_{iD}. In addition, recall that there are four types of scheduling events (arrival, finishing, Zexpiring, Dexpiring) that can require a context switch (here, we consider that the instant when a jobs normal execution is killed and the enforcement execution arrives as a context switch). Thus, in the schedule in the time interval [0,D_{iD}], there are at most $4 \cdot (\sum_{j \in \mathrm{hep(iD)}} \lceil D_{\mathrm{iD}}/T_j \rceil)$ context switches. We represent the time interval [0,D_{iD}] as a set of time intervals such that in each time interval, there is no context switch; we refer to each such time interval as a *position*. We let npos (meaning number of positions) be the sufficient number of positions needed to represent the schedule based on the above upper bound on the number of context switches; thus:

$$\mathrm{npos} = 4 \cdot (\sum_{j \in \mathrm{hep(iD)}} \lceil D_{\mathrm{iD}}/T_j \rceil) - 1 \quad (0)$$

Recall that the number of jobs of a task τ_j with $j \in \mathrm{hp(iD)}$ is at most $\lceil D_{\mathrm{iD}}/T_j \rceil$. All of these jobs must (as stated by (0)) arrive at time 0 or later. But we permit that some of these jobs may arrive after D_{iD}; these jobs, however, are not (from (0)) allowed to perform any execution (they can have normal execution time being zero). We will describe the schedule with variables that are either real, integers, or Booleans. We will use a real variable to indicate the time when an event occurs (for example a job arrival). We will use integer variables to indicate the position at which an event occurs. We will use Boolean variables to indicate whether a certain event occurs in a given position. Recall that there are five possible outcomes for a job (skip, finish in normal mode and increment counter, finish in normal mode and not increment counter, finish in enforcement mode, kill at time of absolute deadline); we will use variables to indicate whether a job has a certain outcome and this outcome was caused by an event in a certain position. We will let j be an index of a task in $\mathrm{hep(iD)}$. We will let q be an index of a job; we will let p be the index of a position.

Variables. We now state the variables, their interpretation, and their domains. The variables with the domain real numbers are the following. t^p denotes the time when position p starts. Note that p in superscript does not mean exponentiation; the superscript is an index. $A_{j,q}$ denotes the arrival time of $\tau_{j,q}$.

$c_{j,q}$ denotes the normal execution time of $\tau_{j,q}$. $e_{j,q}$ denotes the enforcement execution time of $\tau_{j,q}$. $\mathrm{execc}^p_{j,q}$ denotes the amount of normal execution that $\tau_{j,q}$ performs in position p. $\mathrm{exece}^p_{j,q}$ denotes the amount of enforcement execution that $\tau_{j,q}$ performs in position p.

The variables with the domain integers are the following. $\mathrm{arrivespos}_{j,q}$ denote the position at which $\tau_{j,q}$ arrives. $\mathrm{op}_{j,q}$ denote the position at which $\tau_{j,q}$ finishes. (Intuitively, op means that there is an outcome of a job and there is a position at which this outcome is determined.) $\mathrm{Zexpirespos}_{j,q}$ denote the position at which $\tau_{j,q}$ Zexpires. $\mathrm{Dexpirespos}_{j,q}$ denote the position at which $\tau_{j,q}$ Dexpires. $\mathrm{counter}^p_j$ denotes the counter of τ_j in the beginning of position p. The 1^{st} position is position 1; however, we let $\mathrm{counter}^0_j$ denote the value of the counter of τ_j just before position 1 (that is, the initial value of the counter). The variables with the domain Boolean are the following. $\mathrm{respC}_{j,q}$ indicates whether $\tau_{j,q}$ is a respectC job. $\mathrm{elig}^p_{j,q}$ indicates whether $\tau_{j,q}$ is a eligible for execution (i.e., has arrived but not finished) in the beginning of position p. $x^p_{j,q}$ indicates whether $\tau_{j,q}$ executes in position p. $o_{j,q}$ is an integer (in $\{1..5\}$) stating that outcome of job $\tau_{j,q}$.

Constraints. We now state the constraints. Clearly, the start time of a position must be no earlier than the start time of its preceding position. Thus:

$$\forall p \in \{1..\mathrm{npos}\}.t^p \leq t^{p+1} \tag{1}$$

Also, the 1^{st} position starts at time 0. Thus:

$$t^1 = 0 \tag{2}$$

Also, the counters must initially be non-negative:

$$\forall j \in \mathrm{hep(iD)}.\mathrm{counter}^0_j \geq 0 \tag{3}$$

A job arrives in the beginning of exactly one position. We use $\mathrm{arrivespos}_{j,q}$ to indicate the position at which the job $\tau_{j,q}$ arrives. It is an integer. Clearly, it must be in the range of position indices, that is, it must be in $\{1..\mathrm{npos}+1\}$. The same applies to finishing times, Zexpiring, and Dexpiring as well. We also know that each job has at least three distinct positions. Therefore, by knowing the job index, we can obtain tighter bound. Thus:

$$\begin{aligned}\forall\langle j,p\rangle\ (j \in \mathrm{hp(iD)} \cup \{\mathrm{iD}\}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}).\\ (3\cdot(q-1)+1 \leq \mathrm{arrivespos}_{j,q}) \wedge\\ (\mathrm{arrivespos}_{j,q} \leq \mathrm{npos}+1-2-3\cdot(\lceil D_{\mathrm{iD}}/T_j\rceil - q)) \wedge\\ (3\cdot(q-1)+2 \leq \mathrm{Zexpirespos}_{j,q}) \wedge\\ (\mathrm{Zexpirespos}_{j,q} \leq \mathrm{npos}+1-1-3\cdot(\lceil D_{\mathrm{iD}}/T_j\rceil - q)) \wedge\\ (3\cdot(q-1)+3 \leq \mathrm{Dexpirespos}_{j,q}) \wedge\\ (\mathrm{Dexpirespos}_{j,q} \leq \mathrm{npos}+1-3\cdot(\lceil D_{\mathrm{iD}}/T_j\rceil - q)) \wedge\\ (3\cdot(q-1)+1 \leq \mathrm{op}_{j,q}) \wedge\\ (\mathrm{op}_{j,q} \leq \mathrm{npos}+1-3\cdot(\lceil D_{\mathrm{iD}}/T_j\rceil - q))\end{aligned} \tag{4}$$

The time of arrival relates to the the position of arrival. Ditto for other events. Hence:

$$\begin{aligned}\forall\langle j,q,p\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}) \wedge (p \in \{1..\mathrm{npos}+1\}).\\ ((\mathrm{arrivespos}_{j,q}=p) \Rightarrow (A_{j,q}=t^p)) \wedge\\ ((\mathrm{Zexpirespos}_{j,q}=p) \Rightarrow (A_{j,q}+Z_i=t^p)) \wedge\\ ((\mathrm{Dexpirespos}_{j,q}=p) \Rightarrow (A_{j,q}+D_i=t^p))\end{aligned} \tag{5}$$

The above constraint gives us an ordering of arrivespos, Zexpirespos, and Dexpirespos within a job. For tasks with $D_j < T_j$, we also obtain that the Dexpirespos of one job precedes the arrivespos of the next job. For the case that $D_j = T_j$, however, it is necessary to state this explicitly. Thus, we have the following constraint:

$$\begin{aligned}\forall\langle j,q\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil - 1\}).\\ \mathrm{Dexpirespos}_{j,q} + 1 \leq \mathrm{arrivespos}_{j,q+1}\end{aligned} \tag{6}$$

There are also some bounds on execution times.

$$\begin{aligned}\forall\langle j,q\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}).\\ (c_{j,q} \geq 0) \wedge (\mathrm{respC}_{j,q} \Rightarrow (c_{j,q} \leq C_j)) \wedge (e_{j,q} \geq 0) \wedge (e_{j,q} \leq E_j)\end{aligned} \tag{7}$$

We also express minimum inter-arrival times as:

$$\forall\langle j,q\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil - 1\}).\ A_{j,q} + T_j \leq A_{j,q+1} \tag{8}$$

Our system model states that for RC_j consecutive jobs of τ_j, at least one is a respectC job. Thus:

$$\begin{aligned}\forall\langle j,q\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil - (\mathrm{RC}_j - 1)\}).\\ \vee_{q' \in \{q..q+\mathrm{RC}_j-1\}} \mathrm{respC}_{j,q'}\end{aligned} \tag{9}$$

We will now express constraints on the schedule based on how the scheduling is done. A job is eligible if it has arrived but not yet finished. Also, since we use fixed-priority preemptive scheduling, it holds that a job executes at a time if it is eligible at this time and no higher-priority jobs are eligible at this time. Thus:

$$\begin{aligned}\forall\langle j,q,p\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}) \wedge (p \in \{1..\mathrm{npos}\}).\\ (\mathrm{elig}^p_{j,q} = ((\mathrm{arrivespos}_{j,q} \leq p) \wedge (p < \mathrm{op}_{j,q}))) \wedge\\ (\mathrm{x}^p_{j,q} = (\mathrm{elig}^p_{j,q} \wedge (\wedge_{j' \in \mathrm{hp(iD}} \wedge_{q' \in \{1..\lceil D_{\mathrm{iD}}/T_{j'}\rceil\}} (\neg \mathrm{elig}^p_{j',q'}))))\end{aligned} \tag{10}$$

Note that in the constraint above, we are not referring to an event but instead to the time interval of the position. Hence, the case $p = \mathrm{npos}+1$ does not need to be considered. We also express the amount of execution of a certain type (normal versus enforcement) of a job in a given position as follows:

$$\begin{aligned}\forall\langle j,q,p\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}) \wedge (p \in \{1..\mathrm{npos}\}).\\ ((\mathrm{x}^p_{j,q} \wedge (p < \mathrm{Zexpirespos}_{j,q})) \Rightarrow (\mathrm{execc}^p_{j,q} = t^{p+1} - t^p)) \wedge\\ (((\neg \mathrm{x}^p_{j,q}) \wedge (p < \mathrm{Zexpirespos}_{j,q})) \Rightarrow (\mathrm{execc}^p_{j,q} = 0)) \wedge\\ ((p \geq \mathrm{Zexpirespos}_{j,q}) \Rightarrow (\mathrm{execc}^p_{j,q} = 0)) \wedge\\ ((\mathrm{x}^p_{j,q} \wedge (p \geq \mathrm{Zexpirespos}_{j,q})) \Rightarrow (\mathrm{exece}^p_{j,q} = t^{p+1} - t^p)) \wedge\\ (((\neg \mathrm{x}^p_{j,q}) \wedge (p \geq \mathrm{Zexpirespos}_{j,q})) \Rightarrow (\mathrm{exece}^p_{j,q} = 0)) \wedge\\ ((p < \mathrm{Zexpirespos}_{j,q}) \Rightarrow (\mathrm{exece}^p_{j,q} = 0))\end{aligned} \tag{11}$$

Recall that $o_{j,q}$ indicates the outcome of job $\tau_{j,q}$—there are five outcomes. Also recall that $\mathrm{op}_{j,q}$ indicates the position in which an event occurs for which this outcome is determined. Clearly, their ranges are as follows:

$$\begin{aligned}\forall\langle j,q\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}).\\ (1 \leq o_{j,q}) \wedge (o_{j,q} \leq 5) \wedge (1 \leq \mathrm{op}_{j,q}) \wedge (\mathrm{op}_{j,q} \leq \mathrm{npos}+1)\end{aligned} \tag{12}$$

Recall that outcome 1 means that the job is skipped. Hence:

$$\begin{aligned}\forall\langle j,q,p\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}) \wedge (p \in \{1..\mathrm{npos}\}).\\ ((o_{j,q}=1) \wedge (\mathrm{op}_{j,q}=p)) \Rightarrow\\ ((\mathrm{arrivespos}_{j,q} \leq p) \wedge (p \leq \mathrm{Zexpirespos}_{j,q}) \wedge (\sum_{p' \in \{1..p-1\}} \mathrm{execc}^{p'}_{j,q} \leq c_{j,q}) \wedge\\ (\mathrm{counter}^{p-1}_j \geq 1) \wedge (\mathrm{counter}^p_j = \mathrm{counter}^{p-1}_j - 1))\end{aligned} \tag{13}$$

We will now describe the four other possible outcomes. For the case $p = 1$, for $o' \in \{2..5\}$ it holds that $((o_{j,q}=o') \wedge (\mathrm{op}_{j,q}=p)) \Rightarrow \mathrm{false}$. We will now describe the constraints for $p \geq 2$. Recall that outcome 2 represents early finishing and the counter is incremented. Thus:

$$\begin{aligned}\forall\langle j,q,p\rangle\ (j \in \mathrm{hep(iD)}) \wedge (q \in \{1..\lceil D_{\mathrm{iD}}/T_j\rceil\}) \wedge (p \in \{2..\mathrm{npos}\}).\\ ((o_{j,q}=2) \wedge (\mathrm{op}_{j,q}=p)) \Rightarrow\\ ((\mathrm{arrivespos}_{j,q} < p) \wedge (p \leq \mathrm{Zexpirespos}_{j,q}) \wedge\\ x^{p-1}_{j,q} \wedge (\sum_{p' \in \{1..p-1\}} \mathrm{execc}^{p'}_{j,q} = c_{j,q}) \wedge\\ (\mathrm{counter}^{p-1}_j < \mathrm{MAXCOUNT}_j) \wedge (\mathrm{counter}^p_j = \mathrm{counter}^{p-1}_j + 1))\end{aligned} \tag{14}$$

Recall that outcome 3 represents early finishing and the counter is not incremented. Thus:

$$\forall\langle j,q,p\rangle\ (j \in \text{hep(iD)}) \wedge (q \in \{1..\lceil D_{\text{iD}}/T_j\rceil\}) \wedge (p \in \{2..\text{npos}\}).\ ((o_{j,q}=3) \wedge (\text{op}_{j,q}=p)) \Rightarrow ((\text{arrivespos}_{j,q}<p) \wedge (p \leq \text{Zexpirespos}_{j,q}) \wedge x_{j,q}^{p-1} \wedge (\sum_{p' \in \{1..p-1\}} \text{execc}_{j,q}^{p'} = c_{j,q}) \wedge (\text{counter}_j^{p-1} \geq \text{MAXCOUNT}_j) \wedge (\text{counter}_j^{p} = \text{counter}_j^{p-1})) \quad (15)$$

Recall that outcome 4 represents late finishing. Thus:

$$\forall\langle j,q,p\rangle\ (j \in \text{hep(iD)}) \wedge (q \in \{1..\lceil D_{\text{iD}}/T_j\rceil\}) \wedge (p \in \{2..\text{npos}\}).\ ((o_{j,q}=4) \wedge (\text{op}_{j,q}=p)) \Rightarrow ((\text{Zexpirespos}_{j,q}<p) \wedge (p < \text{Dexpirespos}_{j,q}) \wedge x_{j,q}^{p-1} \wedge (\sum_{p' \in \{1..p-1\}} \text{execc}_{j,q}^{p'} < c_{j,q}) \wedge (\sum_{p' \in \{1..p-1\}} \text{exece}_{j,q}^{p'} = e_{j,q}) \wedge (\text{counter}_j^{p} = \text{counter}_j^{p-1})) \quad (16)$$

Recall that outcome 5 represents finishing at deadline and the job gets killed. Thus:

$$\forall\langle j,q,p\rangle\ (j \in \text{hep(iD)}) \wedge (q \in \{1..\lceil D_{\text{iD}}/T_j\rceil\}) \wedge (p \in \{2..\text{npos}\}).\ ((o_{j,q}=5) \wedge (\text{op}_{j,q}=p)) \Rightarrow ((p = \text{Dexpirespos}_{j,q}) \wedge (\sum_{p' \in \{1..p-1\}} \text{execc}_{j,q}^{p'} < c_{j,q}) \wedge (\sum_{p' \in \{1..p-1\}} \text{exece}_{j,q}^{p'} < e_{j,q}) \wedge (\text{counter}_j^{p} = \text{counter}_j^{p-1} - 1)) \quad (17)$$

If no event that creates one of these outcomes occurs, then the counter of a task should be unchanged. Thus:

$$\forall\langle j,q,p\rangle\ (j \in \text{hep(iD)}) \wedge (q \in \{1..\lceil D_{\text{iD}}/T_j\rceil\}) \wedge (p \in \{1..\text{npos}+1\}).\ (\neg(\text{op}_{j,q}=p)) \Rightarrow (\text{counter}_j^{p} = \text{counter}_j^{p-1}) \quad (18)$$

From (0), we obtain:

$$\forall\langle j,p\rangle\ (j \in \text{hp(iD)}) \wedge (p \in \{1..\text{npos}+1\}).\ \text{counter}_j^{p} \geq 0 \quad (19)$$

From (0) it follows that $\tau_{\text{iD},1}$ misses its deadline. Thus:

$$o_{\text{iD},1} = 5 \quad (20)$$

From (0) it follows that the initial value of the counter of τ_{iD} is zero. Thus:

$$\text{counter}_{\text{iD}}^{0} = 0 \quad (21)$$

Recall that (0) states that the processor is busy from time zero until the absolute deadline of $\tau_{iD,1}$. We express this as:

$$\forall p \in \{1..\text{npos}\}.\ (p < \text{Dexpirespos}_{\text{iD},1}) \Rightarrow (\vee_{j \in \text{hep(iD)}} \vee_{q \in \{1..\lceil D_{\text{iD}}/T_j\rceil\}} x_{j,q}^{p}) \quad (22)$$

From (0), we obtain that the processor is idle after this time. Hence:

$$\forall p \in \{1..\text{npos}\}.\ (p \geq \text{Dexpirespos}_{\text{iD},1}) \Rightarrow (\wedge_{j \in \text{hep(iD)}} \wedge_{q \in \{1..\lceil D_{\text{iD}}/T_j\rceil\}} ((\text{execc}_{j,q}^{p}=0) \wedge (\text{exece}_{j,q}^{p}=0))) \quad (23)$$

Note that the two recently stated constraint are asymmetric; one uses x and the other uses execc and exece. The reason for this is our schedule representation assumes that a τ_j generates exactly $\lceil D_{\text{iD}}/T_j\rceil$ jobs but (0) states that τ_j generates at most $\lceil D_{\text{iD}}/T_j\rceil$ jobs; we deal with this (as mentioned earlier) by allowing some of the jobs τ_j to have $c_{j,q}=0$ and $e_{j,q}=0$. These jobs may have x set to true for positions greater than or equal to $\text{Dexpirespos}_{\text{iD},1}$ but then in such positions, their execution is zero (because the position duration is zero).

Further, from (0) it follows that:

$$\forall\langle j,p\rangle\ (j \in \text{hep(iD)}) \wedge (p \in \{1..\text{npos}\}).\ \text{counter}_j^{p} \leq \min(\lceil D_{\text{iD}}/T_j\rceil, \text{MAXCOUNT}_j) \quad (24)$$

From (0) we also obtain the arrival time τ_{iD}. Hence:

$$(A_{\text{iD},1}=0) \wedge (\text{arrivespos}_{\text{iD},1}=1) \quad (25)$$

From (0) we also obtain that the counter of τ_{iD} before its deadline is zero. Hence:

$$\forall p \in \{1..\text{npos}+1\}.\ (p < \text{Dexpirespos}_{\text{iD},1}) \Rightarrow (\text{counter}_{\text{iD}}^{p}=0) \quad (26)$$

And after, it is -1. Hence:

$$\forall p \in \{1..\text{npos}+1\}.\ (p \geq \text{Dexpirespos}_{\text{iD},1}) \Rightarrow (\text{counter}_{\text{iD}}^{p}=-1) \quad (27)$$

We let cprsmtinstance(τ,iD) denote a function that takes a taskset τ and an integer iD as input and outputs an SMT instance; this SMT instance is constructed as given in this subsection. We let sat denote a function that takes an SMT instance as input and outputs true if this SMT instance is satisfiable; otherwise it outputs false.

C. Creating a schedulability test

Lemma IV.1. *τ is not schedulable $\Rightarrow$ (0)*

Proof. Follows from the discussion in Section 4.1. □

Lemma IV.2. *(0)* $\Rightarrow (\exists \tau_{\text{iD}} \in \tau\ \ \text{sat}(\text{cprsmtinstance}(\tau, \text{iD})))$

Proof. Follows from the discussion in Section 4.2. □

Lemma IV.3. *τ is not schedulable* $\Rightarrow (\exists \tau_{\text{iD}} \in \tau\ \ \text{sat}(\text{cprsmtinstance}(\tau, \text{iD})))$

Proof. Follows from Lemma 1 and Lemma 2. □

Lemma IV.4. *τ is not schedulable* $\Leftarrow (\exists \tau_{\text{iD}} \in \tau\ \ \text{sat}(\text{cprsmtinstance}(\tau, \text{iD})))$

Proof. If the rhs is true, then $\exists \tau_{\text{iD}} \in \tau\ \ \text{sat}(\text{cprsmtinstance}(\tau, \text{iD}))$. Then we can obtain a solution to this SMT instance and this yields an assignment R, a schedule sc, and a counter assignment ca such that R is legal, ca is legal, and sc can be generated by R and in which the counter of τ_{iD} is initially zero and the 1^{st} job of τ_{iD} misses its deadline. Hence, the lhs is true. □

Lemma IV.5. *τ is not schedulable* $\Leftrightarrow (\exists \tau_{\text{iD}} \in \tau\ \ \text{sat}(\text{cprsmtinstance}(\tau, \text{iD})))$

Proof. Follows from Lemma 3 and Lemma 4. □

Theorem IV.6. *τ is schedulable* $\Leftrightarrow (\forall \tau_{\text{iD}} \in \tau\ \ \neg\text{sat}(\text{cprsmtinstance}(\tau, \text{iD})))$

Proof. Follows from Lemma 5. □

Based on Theorem 6, we can now create an algorithm for schedulability testing as follows:

1) flag := true
2) **for** each $\tau_{\text{iD}} \in \tau$ as long as flag is true **do**
3) **if** sat(cprsmtinstance(τ, iD)) **then**
4) flag := false
5) **end if**
6) **end for**
7) return flag

V. Tool and Evaluation

We have implemented a tool that performs the schedulability test presented in the previous section. This tool is a C program that generates a file with the SMT instance and invokes Z3—a state-of-the-art SMT solver. If the tool outputs unschedulable for a taskset, then the tool produces a Gantt chart which shows a schedule; this can be used to explain why the taskset is unschedulable. Here, we present experimental results on this tool.

Experimental setup. We generate tasksets using two taskset-generation parameters targetutil, TMAXEXP as follows: $|\tau| = 2$, $T_i = 2^{\mathrm{random}(0.0,\log_2 \mathrm{TMAXEXP})}$, $D_i = T_i$, $Z_i = 0.8 \cdot D_i$, $C_i = 0.9 \cdot Z_i$, $E_i = 0.1 \cdot Z_i$, $\mathrm{RC}_i = 1$, $\mathrm{MAXCOUNT}_i = 1$. After all tasks have been assigned parameters this way, we compute scalingfactor as follows scalingfactor := targetutil/$(\sum_{\tau_j \in \tau} C_j/T_j)$. Then, for each task $\tau_j \in \tau$, we multiply C_j and E_j by scalingfactor. Hence, for the resulting taskset, it will holds that $\sum_{\tau_j \in \tau} C_j/T_j$ = targetutil.

The domains of the taskset-generation parameters are as follows: targetutil $\in \{$ 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0 $\}$, and TMAXEXP $\in \{$ 1, 2, 4, 8, 16, 32, 64, 128 $\}$. We explore all combinations of them; that is 90 combinations. For each combination, we generate one taskset and measure the time required for our tool to perform schedulability analysis.

Experimental results. We found that in our evaluation (with tasksets with two tasks), it never took more than 54 seconds for our schedulability test to finish.

Case study. In our previous work [1], we have studied enforcers, i.e., software components that monitor and can alter the behavior of other software components. As part of this, we built a multi-UAV system in our indoor drone-lab (an area 10x10m). In this system, there are two quad-copter UAVs and there are two enforcers: one enforcer should ensure that the distance between the quad-copters are separated by at least a pre-specified number and the other enforcer should ensure that the geographical location of a UAV is within a given rectangle. Our system uses a logical enforcer to ensure the aforementioned properties and this is achieved by using an SMT solver at run-time to decide if an enforcer should override the application controller. The execution time of the SMT solver at run-time is very variable however. Thus, in some cases (when the execution time is very large), we need to terminate the SMT solver at run-time and then invoke another safe action (pre-computed action that can be taken without getting the result from the SMT solver). It can be seen that this is a very good match to the task model that we present in this paper (indeed, the task model in this paper was inspired by the work in [1]). Therefore, create a model loosely inspired by one of the computers in the multi-UAV system in [1] using the task model in this paper. Table II shows it. We apply our new schedulability test on this taskset and find that it is schedulable.

Task	prio	T	D	Z	C	E	MAXCOUNT	RC
τ_1	2	20	20	10	9	1	2	2
τ_2	1	25	25	12	11	1	2	2

TABLE II
A model of one of the computers in the multi-UAV system in [1].

VI. Conclusions

We presented a new model that describes the current state of the physical environment in terms of how tolerant it is to disruption of the software system and how the scheduling impacts this tolerance; we call this model *Cyber-Physical Resilience* (CPR). For this model, we presented an exact schedulability test and a tool that implements this schedulability test. We evaluated it on randomly-generated tasksets and on a model of a multi-UAV system from [1].

Acknowledgment

Copyright 2018 IEEE. All Rights Reserved. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. DM18-0522

References

[1] B. Andersson, S. Chaki, and D. de Niz. Combining symbolic runtime enforcers for cyber-physical systems. In *RV*, 2017.
[2] A. Anta and P. Tabuada. On the benefits of relaxing the periodicity assumption for networked control systems over CAN. In *RTSS*, 2009.
[3] K.E. Årzén. A simple event-based PID controller. In *IFAC World Congress*, 1999.
[4] G. C. Buttazzo, E. Bini, and D. Buttle. Rate-adaptive tasks: Model, analysis, and design issues. In *DATE*, 2014.
[5] M. Caccamo and G. Buttazzo. Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In *RTSS*, 1997.
[6] M. Clark, X. Koutsoukos, R. Kumar, I. Lee, G. Pappas, L. Pike, J. Porter, and O. Sokolsky. A study on run time assurance for complex cyber physical systems. In *Technical Report, Air Force Research Laboratory*, 2013.
[7] R. I. Davis, T. Feld, V. Pollex, and F. Slomka. Schedulability tests for tasks with variable rate-dependent behaviour under fixed priority scheduling. In *RTAS*, 2014.
[8] J. Kim, K. Lakshmanan, and R. Rajkumar. Rhythmic tasks: A new task model with continually varying periods for cyber-physical systems. In *ICCPS*, 2012.
[9] P. Kumar and L. Thiele. Quantifying the effect of rare timing events with settling-time and overshoot. In *RTSS*, 2012.
[10] C. Lee, C.-S. Shieh, and L. Sha. Online QoS optimization using service classes in surveillance radar systems. In *JRTS*, 2004.
[11] M. Lemmon, T. Chantem, X. Hu, and M. Zyskowski. On self-triggered full information H-infinity controllers. In *Hybrid Systems: Computation and Control*, 2007.
[12] W. Lucia, B. Sinopoli, and G. Franzè. A set-theoretic approach for secure and resilient control of cyber-physical systems subject to false data injection attacks. In *SOSCYPS*, 2016.
[13] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin. On task schedulability in real-time control systems. In *RTSS*, 1996.
[14] L. Sha. Using simplicity to control complexity. *IEEE Software*, 2001.
[15] L. Sha, R. Rajkumar, and S. S. Sathaye. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. In *proceedings of the IEEE*, 1994.
[16] M. Velasco, J. Fuertes, and P. Martí. The self triggered task model for real-time control systems. In *RTSS-WIP*, 2003.
[17] M. Velasco, P. Martí, and E. Bini. Control-driven tasks: Modeling and analysis. In *RTSS*, 2009.