Fall 9-19-2024

A Limited-Preemption Scheduling Model Inspired by Security Considerations

Benjamin Standaert
Washington University in St. Louis, b.g.standaert@wustl.edu

Fatima Raadia
Wayne State University, fatima.fr@wayne.edu

Marion Sudvarg
Washington University in St. Louis, msudvarg@wustl.edu

Sanjoy Baruah
Washington University in St. Louis, baruah@wustl.edu

Thidapat Chantem
Virginia Tech, tchantem@vt.edu

See next page for additional authors

A Limited-Preemption Scheduling Model Inspired by Security Considerations

Benjamin Standaert
Washington University in St. Louis, United States

Fatima Raadia
Wayne State University, United States

Marion Sudvarg[1]
Washington University in St. Louis, United States

Sanjoy Baruah
Washington University in St. Louis, United States

Thidapat Chantem
Virginia Tech, United States

Nathan Fisher
Wayne State University, United States

Christopher Gill
Washington University in St. Louis, United States

Abstract

Safety-critical embedded systems such as autonomous vehicles typically have only very limited computational capabilities on board that must be carefully managed to provide required enhanced functionalities. As these systems become more complex and inter-connected, some parts may need to be secured to prevent unauthorized access, or isolated to ensure correctness.

We propose the *multi-phase secure* (MPS) task model as a natural extension of the widely used sporadic task model for modeling both the timing and the security (and isolation) requirements for such systems. Under MPS, task phases reflect execution using different security mechanisms which each have associated execution time costs for startup and teardown. We develop corresponding limited-preemption scheduling algorithms and associated pseudo-polynomial schedulability tests for constrained-deadline MPS tasks; evaluation shows that these are efficient to compute for bounded utilizations. We empirically demonstrate that the MPS model successfully schedules more task sets compared to non-preemptive approaches.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases real-time systems, limited-preemption scheduling, trusted execution environments

Digital Object Identifier 10.4230/LITES.1.1.42

Acknowledgements This research was supported in part by NSF Grants CPS-1932530, CNS-2141256, CNS-2229290, IIS-1724227, CNS-2038609, CCF-2118202, CNS-2211641 and CPS-2038726. We, the authors, would like to thank the reviewers for their constructive remarks.

Received Date of submission **Accepted** Date of acceptance **Published** Date of publishing

Editor LITES section area editor

1 Introduction

In today's interconnected world, the security of real-time systems has emerged as a primary concern, e.g., [27, 18, 20, 17, 26, 29, 22], given the widespread integration of electronic devices into

[1] Corresponding author

various aspects of daily life. However, the implementation of security measures often introduces additional resource requirements, such as increased computational overhead, or imposes specific constraints on application behaviors; for example, this could involve necessitating computation that requires isolation or cannot be preempted.

For example, control flow integrity (CFI) checks may be needed to ensure correct program execution. However, such checks, which require CPU time in addition to normal code execution, must be carried out at specific time points (e.g., after branching) and allowing for preemption may result in an arbitrary computation being performed but not detected. As another example, a task that is responsible for taking sensor readings may need to execute in isolation in order to to ensure that another task cannot deduce when an event of interest occurs [22].

Since implementing security measures requires some of the same resources that the real-time tasks need to advance their execution, a co-design approach that explicitly considers security cost/requirements along with real-time requirements is potentially more effective at managing limited computational resources. For instance, *trusted execution environments* (TEEs) provide isolation of code and data in hardware at the expense of startup and teardown costs (in the order of microseconds for Arm Cortex-M [21] and hundreds of microseconds or even tens of milliseconds for Arm Cortex-A [23]). A scheduling approach that does not consider this specific security-driven overhead may elect to switch between the secure world (i.e., executing in TEE) and the normal world (no TEE) indiscriminately. This may result in an excessive amount of overhead and cause deadline misses. A security-cognizant scheduler, on the other hand, would make judicious decisions based on both security and real-time requirements, e.g., by bundling up multiple TEE executions and executing them one immediately after the other so as to have to pay for startup and teardown cost only once [23].

A recent ISORC paper [6] proposed and developed algorithms that are able to provide provable correctness of both the timing and some security properties. We believe that such a scheduling-based approach to achieving security in safety-critical systems is possible, and indeed, necessary in embedded systems that are particularly cost- and SWaP-constrained and hence need to be implemented in a resource-efficient manner. However, we consider it unlikely that a ‘one size fits all’ solution exists; instead, security-cognizant scheduling must first explicitly identify the kinds of threats that are of concern by precisely defining a threat model, and design scheduling strategies that can be proved to be resistant to attacks under the identified threat model. We consider the research in [24] to be particularly noteworthy in this regard, in their explicit and methodical modeling of different threats in the context of the sporadic task model, and their analysis of vulnerabilities of current strategies (including security-agnostic fixed-priority scheduling [1] and the randomization-based schedule obfuscation approaches, e.g., the one in [30]) to such threats.

Security-Cognizant Scheduling. We believe the methodology formalized and used in [24] holds great promise as a means of integrating security and timing correctness concerns within a common framework. This methodology was articulated in [6] as follows: security for safety-critical real-time embedded systems can be achieved by *(i)* explicitly representing specific security considerations within the same formal frameworks that are currently used for specifying real-time workloads, thereby extending notions of correctness to incorporate both the timing and the security aspects; and *(ii)* extending previously-developed techniques for achieving provable timing correctness to these models, thus assuring that both timing and security properties are correct.

This Work. This paper **extends our prior work** in [5], which applied the methodology articulated in [6] to the following problem in system design for real-time + security. We consider computer platforms upon which *multiple different security mechanisms* (such as TEEs, encryption/decryption co-processors, FPGA-implemented secure computations, etc.) co-exist. Depending upon their security requirements, different pieces/parts of the (real-time) code may need to use different security mechanisms at different times. We therefore assume that the code is broken up into *phases*, with different consecutive phases needing to use different security mechanisms — the security mechanism used by each phase is specified for the phase. We assume that there is a startup/teardown overhead cost (for data communication, initialization, etc.) expressed as an execution duration, associated with switching between different security mechanisms. I.e., *there is a time overhead associated with switching between the execution of different phases.*

Contributions. As in our prior work [5], we formalize the workload model discussed above as the *Multi-Phase Secure* (**MPS**) task model, with multiple independent recurrent processes of this kind that are to execute upon a single shared preemptive processor. We start out in Section 4 assuming that each recurrent process is represented using the widely-used 3-parameter sporadic task model [3]; later in Section 5, we will consider a generalization that models conditional execution within each recurrent process. For both models, we represent the problem of ensuring timeliness plus security as a schedulability analysis problem, which we then solve by adapting results obtained in prior work (e.g., [4, 8, 10, 25, 11]) on limited-preemption scheduling.

This paper **extends, corrects, and clarifies** our prior results in [5]. Sections 4 and 5 introduce pseudo-polynomial schedulability analysis for sets of MPS tasks, improving the execution time of the associated algorithms, especially for tasks with implicit deadlines. With these improvements, we are able to evaluate larger sets of tasks with more realistic ranges of periods. We also address inconsistencies in [5] between the theoretical model and its implementation by using a continuous-time representation of the algorithms. These are reflected in the new results in Section 6, which more clearly demonstrate the advantages of our approach over non-preemptive schedulers.

We emphasize that although the design of these models are motivated by security considerations – they arose out of some security-related projects that we are currently working on – we are proposing a *scheduling model* and associated algorithms, not a complete solution to a particular security problem. That is, although our model draws inspiration from security concerns, it (i) does not claim a perfect match to all security requirements; and (ii) it should have applicability beyond the security domain – indeed, we suggest that the results presented in this paper be looked upon as a generalization of the rich body of real-time scheduling theory literature on limited-preemption scheduling.

Organization. The remainder of this manuscript is organized as follows. After briefly discussing some related scheduling-theory results in Section 2, we formally define the MPS sporadic tasks model in Section 4, and provide both pre-runtime analysis and a run-time scheduling algorithm for MPS sporadic task systems upon preemptive uniprocessor platforms. In Section 5 we further generalize the workload model to be able to represent conditional execution, and extend the algorithms of Sec. 4 to become applicable to this more general workload model as well. We have performed some schedulability experiments to evaluate the effectiveness of our algorithms – we report on these experiments in Section 6. We conclude in Section 7 by pointing out some directions in which we intend to extend this work, and by placing our results within a larger context on the timing- and security-aware synthesis of safety-critical systems.

2 Some Real-Time Scheduling Background

2.1 The Sporadic Task Model [3]

In this model, recurrent processes are represented as sporadic tasks $\tau_i = (C_i, D_i, T_i)$. Each task has three defining characteristics: worst-case execution requirement (WCET) C_i, relative deadline D_i, and period (minimum inter-arrival duration) T_i. The sporadic task τ_i generates a series of jobs, with inter-arrival times of at least T_i. Each job must be completed within a scheduling window, which starts at the job's release time and ends D_i time units later, and the job's execution time is limited to C_i units. A sporadic task system Γ is made up of multiple independent sporadic tasks. We assume without loss of generality that tasks are indexed in non-decreasing relative deadline order (i.e., if $i < j$ then $D_i \leq D_j$).

Processor Demand Analysis (PDA). A sporadic task system can be scheduled optimally by the Earliest Deadline First (EDF) [19] scheduling algorithm, given a preemptive uniprocessor. To determine whether a specific task system can be correctly scheduled by EDF, Processor Demand Analysis (PDA) [7] can be utilized. PDA is a necessary and sufficient algorithm that is also optimal. The key idea of PDA is built upon the *demand bound function* (DBF). Given an interval length of L such that $L \geq 0$, the DBF for a sporadic task τ_i can be represented by $\text{DBF}_i(L)$: the maximum possible aggregate execution time required by jobs of task τ_i such that they arrive in L and have deadlines before L. The following equation was derived in [3] to compute its value:

$$\text{DBF}_i(L) = \max\left(\left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1, 0\right) \times C_i \tag{1}$$

For a task system τ to be correctly scheduled by EDF, the following was derived in [3] as a necessary and sufficient condition for all $L \geq 0$:

$$\left[\left(\sum_{\tau_i \in \Gamma} \text{DBF}_i(L)\right) \leq L\right] \tag{2}$$

The Testing Set. A naïve application of PDA requires testing the validity of Equation 2 for all intervals. However, a more efficient approach, outlined in [3], involves checking only values of L that follow the pattern $L \equiv (k \times T_i + D_i)$ for some non-negative integer k and some $\tau_i \in \Gamma$.

Furthermore, it suffices to test such values that are less than the least common multiple of all the T_i parameters. The collection of all such values of t for which it's necessary to verify that Condition 2 holds true in order to confirm EDF-schedulability is referred to as the *testing set* for the sporadic task system Γ, often denoted as $\mathcal{T}(\Gamma)$.

It is worth noting [3] that, in general, the size $|\mathcal{T}(\Gamma)|$ of the testing set $\mathcal{T}(\Gamma)$ can be exponential in the representation of τ. However, it has been proven [2, Theorem 3.1] that for *bounded-utilization* task systems —i.e., systems Γ that fulfill the additional requirement that $\sum_{\tau_i \in \Gamma} U_i \leq c$ for some fixed constant c strictly less than 1— it is sufficient to check a smaller testing set with pseudo-polynomial cardinality relative to the representation of Γ, consisting of all values of the form $L \equiv (k \times T_i + D_i)$ not exceeding

$$\min\left(P, \max\left(D_{\max}, \frac{1}{1-U} \cdot \sum_{i=1}^{n} U_i \cdot (T_i - D_i)\right)\right) \tag{3}$$

where P is the least common multiple of all T_i, and $D_{\max}$ is the maximum of all D_i parameters. We note that for bounded-utilization *implicit-deadline* tasks —those for which $T_i = D_i$ for every task τ_i— this bound reduces to $D_{\max}$. In this extension, we apply this smaller testing set to our scheduling algorithms for MPS tasks.

Since we can check Condition 2 in linear ($\Theta(n)$) time for any given value of t, these observations imply that we can perform an exponential-time EDF-schedulability test for general task systems and a pseudo-polynomial-time one for bounded-utilization systems.

Unfortunately, the general problem is NP-hard in the strong sense [12, 15, 14], and the bounded-utilization variant is NP-hard in the ordinary sense [13]. Therefore, it's unlikely that we will discover more efficient schedulability tests.

2.2 Limited-Preemption Scheduling

The limited-preemption sporadic task model, as introduced by Baruah et al. [4], adds to the task specification $\tau_i = (C_i, D_i, T_i, \beta_i)$ a *chunk-size* parameter β_i in addition to the regular parameters C_i, D_i, and T_i. This parameter β_i indicates that each job of task τ_i may need to execute non-preemptively for up to β_i time units.

To schedule tasks in the limited-preemption sporadic task model, the limited-preemption EDF scheduling algorithm was proposed [4, 8]. Like its preemptive counterpart, the limited-preemption EDF algorithm prioritizes jobs based on their (absolute) deadlines. If a job of task τ_i with remaining execution time e is executing and a new job with an earlier deadline arrives, then τ_i's job may execute for an additional $\min(e, \beta_i)$ time units before incurring a preemption.

Baruah and Bertogna [4, 8] showed that a task system is *not* schedulable under the limited-preemption EDF model if and only if either of the following conditions are true:

$$\exists L : L \geq 0 : \sum_{\tau_i \in \Gamma} \text{DBF}_i(L) > L \tag{4}$$

or

$$\exists \tau_i : \exists L : 0 \leq L < D_i : \beta_i + \sum_{\tau_j \in \Gamma, j \neq i} \text{DBF}_j(L) > L \tag{5}$$

Noting that $\text{DBF}_i(L) = 0$ when $L < D_i$, we combine and invert the two conditions, giving a necessary and sufficient condition for successfully scheduling a limited-preemption sporadic task system Γ upon a single preemptive processor using the limited-preemption EDF algorithm:

$$\forall L, \quad \left[\left(\sum_{\tau_i \in \Gamma} \text{DBF}_i(L) \right) + \overbrace{\max_{\{\tau_i | D_i > L\}} \left\{ \beta_i \right\}}^{\text{(blocking due to limited preemption)}} \leq L \right] \tag{6}$$

Unlike the exact test for preemptive uniprocessor EDF-schedulability (Equation 2), Equation 6 contains an additional term on the left-hand side of the inequality that accounts for blocking due to later-deadline (and hence lower-priority) jobs. Specifically, the $\max_{\tau_i | D_i > L} \beta_i$ term is a *blocking term* that captures the potential delay caused by lower-priority jobs that were already executing at the start of the interval, for a duration of up to their chunk size. Since we assume that all tasks have non-negative execution time, this blocking term is always non-negative.

In this extension to the prior work in [5], we use a continuous-time representation of the blocking term in Equation 6. The prior work used a discrete-time representation, $\max_{\tau_i | D_i > L} \beta_i - 1$, which requires both task periods *and execution times* to be represented as integers. This introduces several challenges. First, it is more difficult to reason about the effect of inserting preemption points; blocking times (chunk sizes) must also be represented as integers, but the number of "chunks" might not evenly divide the execution time. Second, there is a tradeoff when choosing the precision at which to represent execution time units. If the unit of time is coarse, then the execution time of each phase and its corresponding startup/teardown time must be rounded up. If the unit of time is very short —such as a single processor tick— to achieve higher precision, then the testing set grows rapidly as the representations of the task periods become larger. By using continuous time, this extension removes these limitations, and modifies the expression to allow execution times to take any non-negative real value.

Based on this observation, the Processor Demand Analysis (PDA) algorithm has been extended to apply to limited-preemption systems as well [4]. The extension is straightforward: Equation 6 replaces Equation 2 in the algorithm, and the algorithm proceeds as usual.

However, we note that Expression 6 cannot hold true for values of L of the form

$$0 \leq L < \min\left(D_{\min}, \max_{\{\tau_i\}}\{\beta_i\}\right) \tag{7}$$

where $D_{\min} = \min_i\{D_i\}$, suggesting **incorrectly** that *a set of tasks is not EDF schedulable if any task has non-zero blocking time.* In the next section of this extension, we present a corrected condition that addresses this issue, which we then apply to scheduling of MPS tasks in Section 4.

3 The Corrected Limited-Preemption EDF Schedulability Condition

In this section, we present a correction to the condition of Baruah and Bertogna [4, 8] for EDF schedulability of limited-preemption tasks.

3.1 The Problem

As discussed in the prior section, in [4, 8], Baruah and Bertogna claimed as a necessary and sufficient condition for scheduling a limited-preemption sporadic task system upon a single preemptive processor using EDF that

$$\forall L, \quad \left[\left(\sum_{\tau_i \in \Gamma} \text{DBF}_i(L)\right) + \overbrace{\max_{\{\tau_i | D_i > L\}}\left\{\beta_i\right\}}^{\text{(blocking due to limited preemption)}} \leq L\right] \tag{8}$$

where β_i is the blocking due to limited preemption induced by task τ_i.

From the definition of the demand bound function $\text{DBF}_i(L)$ in Equation 1, we observe that $\text{DBF}_i(L) = 0$ for $L < D_i$. Then for $L < D_{\min}$ (i.e., $L < D_i$ for all tasks τ_i), the above condition requires $L \geq \max_{\tau_i | D_i > L}\{\beta_i\}$; as we are already considering the case that $L < D_{min}$, this can be simplified to $L \geq \max_{\tau_i}\{\beta_i\}$.

This implies that for values of L such that $L < D_{\min}$, the above condition cannot hold true if L does not also exceed β_i for all tasks τ_i. More simply, the condition cannot hold true for values of L of the form shown in Expression 7. Because processor demand analysis requires that the condition hold true for *all* values of $L \geq 0$, this would deem **any set of limited-preemption tasks with non-zero blocking time to be unschedulable by EDF.**

3.2 The Correction

As this is obviously not true – i.e., there **are** limited-preemption task sets that are schedulable by EDF, we now set out to correct the above condition. We do so by pointing out a subtlety to the **proof** in [4] of the condition in Expression 6.

That proof constructs the blocking time condition by defining a minimal unschedulable set of jobs for which t_a represents the earliest arrival time of those jobs and t_f is the time at which the first deadline miss occurs. In this system, there is exactly one job with a deadline after t_f; we let τ_j be the task that generates this job. If t_1 is the time at which that job begins to execute non-preemptively and t_2 is when it stops executing, then [4, Equation 4] states that

$$\sum_{i=1, i \neq j}^{n} \text{DBF}(\tau_i, t_f - t_1) > t_f - t_2$$

The proof in [4] then claims that $t_2 - t_1 \leq q_j$, since q_j is an upper bound on the nonpreemptive execution time. We observe that since $t_2 \leq t_f$, **it also follows that** $t_2 - t_1 \leq t_f - t_1$. We can therefore combine these inequalities to make the statement that $t_2 - t_1 \leq \min(q_j, t_f - t_1)$. In light of this, we modify the rest of the proof in [4]. Now, it follows that

$$\sum_{i=1, i\neq j}^{n} \text{DBF}(\tau_i, t_f - t_1) + \min(q_j, t_f - t_1) > t_f - t_2 + (t_2 - t_1)$$

We then replace the expression $t_f - t_1$ with a time L. Since $t_f < D_j$, it follows that $L < D_j$; the DBF of τ_j will therefore be zero at L and we can rewrite the condition as:

$$\sum_{i=1}^{n} \text{DBF}(\tau_i, L) + \min(q_j, L) > L$$

This condition checks whether some task τ_j causes excessive blocking at times $L < D_j$; to determine if the system is schedulable, we can therefore test whether the following condition holds for any task τ_i at any time $L \geq 0$, considering blocking times from just those tasks for which $L < D_i$:

$$\left(\sum_{\tau_i \in \Gamma} \text{DBF}_i(L)\right) + \min\left(L, \max_{\{\tau_i | D_i > L\}} \{\beta_i\}\right) \leq L \qquad (9)$$

Then when $L < D_{\min}$, the DBF for each task is 0, so the condition will always be satisfied. Furthermore, when $L \geq D_{\min}$, $\sum_i \text{DBF}(L_i) > 0$, and so the condition cannot be satisfied when $\min(L, \max_i \beta_i) \geq L$. We can therefore begin the testing set of our implementation at $D_{\min}$, and only check the blocking time when determining whether the condition is violated.

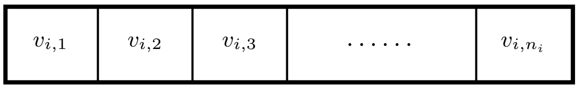
4 Systems of Multi-Phase Secure (MPS) Sporadic Tasks

In this section we extend the sporadic task model to consider the setting where the workload of each task comprises an ordered sequence of different *phases*, with each phase required to use a different security mechanism. Therefore, switching between phases incurs some *teardown/startup overhead*, which translates to an additional execution duration. We are given a system of multiple such independent tasks that are to be scheduled upon a shared preemptive processor. Note that if an executing task is preempted within a phase, the assumption that the tasks in the system are independent of one another implies that we must conservatively assume that the preempting task may be executing using a different security mechanism; hence, the teardown/startup overhead may be incurred again. When a phase of a task is selected for execution we assign it responsibility for taking care of both the startup that must happen at that point in time, and the subsequent teardown that occurs when it either completes execution or is preempted by a higher-priority task. Hence a phase with execution duration c that is preempted k times is responsible for (and hence should have been budgeted for) a total execution duration of

$$c + (k+1) \times (\text{startup cost} + \text{teardown cost}) \qquad (10)$$

4.1 Task Model

We have a task system Γ comprising N independent recurrent tasks $\tau_1, \tau_2, \ldots, \tau_N$, to be scheduled upon a single preemptive processor. The task τ_i is characterized by a period/inter-arrival separation parameter T_i and a relative deadline $D_i \leq T_i$. The body of the task – the work that must be executed each time the task is invoked – comprises n_i *phases*, denoted $v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}$, that must execute in sequence upon each job release of the task:



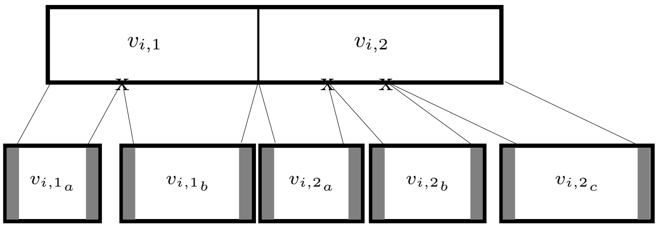
As previously discussed, we assume that successive phases are required to execute using different security mechanisms (i.e., $v_{i,j}$ and $v_{i,j+1}$ execute using different security mechanisms for all j, $1 \le j < n_i$). Let $c(v_{i,j})$ denote the WCET of phase $v_{i,j}$, and $q(v_{i,j})$ denote the sum of the startup cost and the teardown cost associated with the security mechanism within which phase $v_{i,j}$ is to execute. The aggregate WCET of all phases of this task during its execution is thus given by the following expression

$$\sum_{j=1}^{n_i} \Big(c(v_{i,j}) + q(v_{i,j}) \Big)$$

However, suppose that during some execution of task τ_i the j'th phase is preempted k_j times for each j, $1 \le j \le n_i$; as discussed above (Equation 10), the cumulative WCET of all phases of this task during this execution is then given by the expression

$$\sum_{j=1}^{n_i} \Big(c(v_{i,j}) + (k_j + 1) \times q(v_{i,j}) \Big)$$

The figure below depicts a 2-phase job, denoted by $v_{i,1}$ and $v_{i,2}$, which is preempted once in the first phase and twice in the second phase. The shaded region denotes the startup and teardown costs for each phase of the job.



4.2 Overview of Approach

Given a task system Γ comprising multiple independent MPS tasks to be scheduled upon a single preemptive processor, we will first execute a schedulability analysis algorithm to determine whether this system is schedulable, i.e., whether we can guarantee to schedule it to always meet all deadlines, despite the costs incurred by startup/teardown. This schedulability analysis algorithm essentially constructs a limited-preemption task $\widehat{\tau_i}$ corresponding to each task τ_i, and determines whether the resulting limited-preemption task system can be scheduled by the limited-preemption EDF scheduling algorithm [4, 8] to always meet all deadlines. If so, then during run-time the original task system is scheduled using the limited-preemption EDF scheduling algorithm, with chunk-sizes as determined for the corresponding constructed limited-preemption tasks. We point out that if a chunk-size β_i is determined for the limited-preemption task $\widehat{\tau_i}$, then the j'th phase of τ_i's jobs will execute in no more than $\lceil c(v_{i,j})/(\beta_i - q(v_{i,j})) \rceil$ contiguous time-intervals (i.e., it would experience at most $(\lceil c(v_{i,j})/(\beta_i - q(v_{i,j})) \rceil - 1)$ preemptions); equivalently, the cumulative WCET of all the phases of each of task τ_i's jobs will be no more than

$$\sum_{j=1}^{n_i} \Big(c(v_{i,j}) + \left\lceil \frac{c(v_{i,j}}{\beta_i - q(v_{i,j})} \right\rceil \times q(v_{i,j}) \Big)$$

Assumption of Fixed-Preemption Points. We make the conservative assumption that once the chunk-size is determined for each task then the preemption points in the code are statically determined prior to run-time. That is, once the chunk-size β_i is determined for a task τ_i, a preemption is statically inserted into the task's code after the code has executed non-preemptively for no more than β_i time units; this is referred to as the *fixed-preemption point model* [28]. Once τ_i's program reaches this statically-placed preemption point, the security mechanism for the task's current phase must

1. complete a teardown (e.g., a flush of the cache, or ending a TEE session) to ensure task execution integrity during preemption;
2. invoke the operating system's scheduler to see if there are any high-priority tasks awaiting execution; and
3. upon resuming execution as the highest-priority task the security mechanism must perform a startup (e.g., starting a new TEE session from the task's last executed instruction).

Since the preemption points are statically inserted into the code, we must perform the teardown/startup for a phase each time a preemption point is encountered (even if there is no other task active in the system at that time). While clearly this approach suffers from potentially performing unnecessary preemptions, it is often used in safety-critical settings due the precise predictability that fixed-preemption points provide. In future work, we will explore the floating preemption point model and other models that would permit the system to avoid unnecessary preemptions and teardowns/startups.

4.3 The Schedulability Test

We now describe our schedulability test. As discussed above, our approach is to construct for each task τ_i a corresponding limited-preemption task $\widehat{\tau_i}$. This limited-preemption task is assigned the same relative deadline parameter value (i.e., D_i) and the same period parameter value (i.e., T_i) as τ_i; its WCET $\widehat{C_i}$ and its chunk-size parameter β_i are computed as described below, and in pseudo-code form in Algorithm 1.

We introduce integer variables $\text{cnt}(v_{i,j})$ for each i, $1 \le i \le n$, and for each j, $1 \le j \le n_i$, to denote the maximum number of contiguous time-intervals in which the j'th phase of τ_i may need to execute. Then $\widehat{C_i}$, the WCET of each job of task $\widehat{\tau_i}$, can be written as

$$\widehat{C_i} \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} \Big(c(v_{i,j}) + \text{cnt}(v_{i,j}) \times q(v_{i,j}) \Big) \tag{11}$$

where the second term within the summation represents the maximum preemption overhead (the startup cost plus the teardown cost) that is incurred by the j'th phase of task τ_i.

It remains to specify the values we will assign to the $\text{cnt}(v_{i,j})$ variables. We will start out assuming that each phase of each task τ_i executes non-preemptively – i.e., in one contiguous time-interval. We do this by initially assigning each $\text{cnt}(v_{i,j})$ the value 1; we will describe below how the $\text{cnt}(v_{i,j})$ values are updated if enforcing such non-preemptive execution may cause deadlines to be missed. With each $\text{cnt}(v_{i,j})$ assigned the value 1, it is evident that the largest duration for which task τ_i will execute non-preemptively is equal to $\max_{j=1}^{n_i} \{c(v_{i,j}) + q(v_{i,j})\}$; we initialize the chunk-size parameters —the β_i values— accordingly: for each task τ_i,

$$\beta_i \leftarrow \max_{j=1}^{n_i} \{c(v_{i,j}) + q(v_{i,j})\} \tag{12}$$

In this manner, we have instantiated the parameters for one limited-preemption task $\widehat{\tau_i}$ corresponding to each task $\tau \in \Gamma$. We must now check whether the limited-preemption task system

Algorithm 1 The Preprocessing Algorithm for Systems of MPS Sporadic Tasks (see Section 4)

Input: (Γ)

for *each task* $\tau_i \in \Gamma$ **do** //Initially, assume that no preemption is needed
  for $j \leftarrow 1$ **to** n_i **do**
    $\text{cnt}(v_{i,j}) \leftarrow 1$
  $\beta_i \leftarrow \max_{1 \le j \le n_i} (c(v_{i,j}) + q(v_{i,j}))$

//The testing set $\mathcal{T}(\Gamma)_1$ is all $t \equiv D_i + k \cdot T_i$, $t \le D_{\max}$ for some task τ_i and some $k \in \mathbb{N}$.
for t_d *iterating in increasing order over* $\mathcal{T}(\Gamma)_1$ **do**
  Compute $\Delta(t_d)$ as per Eqn 13 //This represents the *slack* in the schedule at t_d
  if $\Delta(t_d) < 0$ **then** //Check whether previously-assigned chunk sizes causes deadline miss
    return THE SYSTEM IS NOT SCHEDULABLE
  for *each* τ_i *for which* $D_i > t_d$ **do**
    if $(\beta_i > \Delta(t_d))$ **then** //Must reduce the value of β_i
      $\beta_i \leftarrow \Delta(t_d)$
      for $j \leftarrow 1$ **to** n_i **do** //For each phase of τ_i, ensure that it doesn't block too much
        if $\beta_i > q(v_{i,j})$ **then** //There is sufficient time in chunk to do task execution
          $\text{cnt}(v_{i,j}) \leftarrow \min_{\kappa \in \mathbb{N}}$ such that $\frac{c(v_{i,j})}{\kappa} + q(v_{i,j}) \le \beta_i$ //Break $c(v_{i_j})$ into small enough pieces
        else
          return THE SYSTEM IS NOT SCHEDULABLE

if *system utilization* > 1 **then**
  return THE SYSTEM IS NOT SCHEDULABLE

if *all tasks have implicit deadlines* $D_i = T_i$ **then**
  return THE SYSTEM IS SCHEDULABLE

//For systems of constrained-deadline tasks, the testing set $\mathcal{T}(\Gamma)_2$ is all $t \equiv D_i + k \cdot T_i$, $D_{\max} < t$ and not exceeding the bound defined in Expression 3 for some task τ_i and some $k \in \mathbb{N}$.
for t_d *iterating in increasing order over the testing set* $\mathcal{T}(\Gamma)_2$ **do**
  Compute $\Delta(t_d)$ as per Eqn 13
  if $\Delta(t_d) < 0$ **then**
    return THE SYSTEM IS NOT SCHEDULABLE

return THE SYSTEM IS SCHEDULABLE

$\widehat{\Gamma} = \{\widehat{\tau_1}, \widehat{\tau_2}, \ldots, \widehat{\tau_N}\}$ so obtained is schedulable using the limited-preemption EDF scheduling algorithm [4, 8]. We do so by checking whether Equation 9 holds for values of t in the testing set $\mathcal{T}(\widehat{\Gamma})$, considered in *increasing order*. First, we split $\mathcal{T}(\widehat{\Gamma})$ into two sets, $\mathcal{T}(\widehat{\Gamma})_1$ containing all values up to and including $D_{\max} \equiv \max_{\tau_i} D_i$, and $\mathcal{T}(\widehat{\Gamma})_2$ containing all values thereafter. Then, we initialize t_d to denote the smallest value in $\mathcal{T}(\widehat{\Gamma})_1$, and perform the following steps.

1. If Equation 9 is satisfied for t_d, we set t_d to be the next-smallest value in $\mathcal{T}(\widehat{\Gamma})_1$, and repeat this step.
2. If Equation 9 is violated for t_d and the first term in the LHS of Equation 9 is $> t_d$, then we conclude that THE SYSTEM IS NOT SCHEDULABLE and return.

Suppose, however, that a violation of Equation 9 occurs due to the blocking term in Equation 9. That is, the first term in the LHS of Equation 9 is $\le t_d$ when Equation 9 is instantiated with $t \leftarrow t_d$, but the sum of the first and second terms exceeds t_d. For this to happen, it must be the case that some $\widehat{\tau}_i$ with $D_i > t_d$ is blocking "too much;" we must reduce the amount of blocking each such task can cause (i.e., reduce its β_i parameter). Below, we describe how to do so.

3. Let $\Delta(t_d)$ denote the amount of blocking that can be tolerated at t_d without causing a deadline miss:

$$\Delta(t_d) \stackrel{\text{def}}{=} t_d - \sum_{\widehat{\tau_k} \in \widehat{\Gamma}} \text{DBF}(\widehat{\tau_k}, t_d) \tag{13}$$

As discussed above, each $\widehat{\tau_i}$ with $D_i > t_d$ must ensure that its blocking term, β_i, is no greater than $\Delta(t_d)$. For each such task with β_i currently greater than $\Delta(t_d)$, we may need to *in*crease $\text{cnt}(v_{i,j})$, the number of contiguous time-intervals in which its j'th phase may execute for each of its phases $v_{i,1}, v_{i,2}, \ldots, v_{i,n_i}$, in the following manner:

$$\text{cnt}(v_{i,j}) \leftarrow \min_{\kappa \in \mathbb{N}} \text{such that} \frac{c(v_{i,j})}{\kappa} + q(v_{i,j}) \leq \Delta(t_d) \tag{14}$$

That is, we reduce blocking in order to satisfy Equation 9 for t_d by potentially increasing the number of preemptions (and thereby incurring additional teardown/startup overhead). In prior work [5], we assumed that tasks had integer execution times and hence used $\beta_i - 1$ as the blocking term; here, we use continuous time and therefore do not subtract a time segment.

4. Such additional overhead must be accounted for; this requires that $\widehat{C_i}$, the WCET parameter of the limited-preemption task $\widehat{\tau_i}$, must be updated (i.e., potentially increased) by recomputing it using Equation 11 (reproduced below):

$$\widehat{C_i} \leftarrow \sum_{j=1}^{n_i} \Big(c(v_{i,j}) + \text{cnt}(v_{i,j}) \times q(v_{i,j}) \Big)$$

(Notice that since some of the $\text{cnt}(v_{i,j})$ values may have increased, the value of $\widehat{C_i}$ may also increase.)

5. Recall that we have been checking the validity of Equation 9 for values of t_d in $\mathcal{T}(\widehat{\Gamma})_1$, the partial testing set of Γ, considered in increasing order. Since we are currently considering t_d, we have therefore already validated that Equation 9 previously held for values of $t < t_d$ in the testing set. The crucial observation now is that *the increase in the value of $\widehat{C_i}$ for any i with $D_i > t_d$ does not invalidate Equation 9 for any $t < t_d$*, because the increased $\widehat{C_i}$ values only contribute to the cumulative demand (the first term in the LHS of Equation 9) for values of $t \geq t_d$. Hence, we do not need to go back and re-validate Equation 9 for values of t smaller than t_d.

6. Having thus modified the $\text{cnt}(v_{i,j})$ variables (as in Equation 14) in order to ensure that Equation 9 is satisfied by $\widehat{\Gamma}$ for t_d, we update the value of t_d to the next-smallest value in $\mathcal{T}(\widehat{\Gamma})_1$, and return to Step 1 above.

7. Once t_d reaches $D_{\max}$, there are no remaining tasks with $D_i > t_d$, and so we are done updating the cnt variables. At this point, we check if the total utilization of the system, $\sum_{i=1}^{n} \frac{WCET(\tau_i)}{T_i} > 1$. If it is, the system cannot be scheduled.

8. For an implicit-deadline system with $U \leq 1$, we prove in Lemma 2 that the slack will never be < 0 at any later time. Therefore, we know that the system is schedulable and can return immediately.

9. For a constrained-deadline system, we check the slack at each remaining point t_d in the testing set $\mathcal{T}(\widehat{\Gamma})_2$ up to and including the testing set's upper bound, described by Expression 3. If the slack is found to be < 0, the system is unschedulable, and we return immediately. Otherwise, is system is deemed to be schedulable.

Computational Complexity. The number of iterations of the *for-loops* of Lines 5 and 21 dominate the computational complexity; the other loops in the algorithm are polynomial in the number of tasks or number of vertices in a chain. The number of iterations of Line 5 is proportional to $D_{\max}$; for implicit-deadline systems, this is the only loop that executes. For constrained-deadline systems, the combined number of iterations for both loops is the number of testing set points. In general, for constrained-deadline sporadic task systems scheduled on a single processor, the testing set can be exponential in the number of tasks, but is psuedo-polynomial as long as the utilization is bounded by a fixed constant strictly less than 1 [2]. For MPS sporadic tasks, the utilization can change as we add preemption points. However, because we only continue to add preemption points as the testing set is traversed up to $D_{\max}$, the testing set remains pseudo-polynomial in size unless the utilization reaches exactly 1, in which case it becomes exponential (bounded by the least-common multiple of the task periods).

Proof of Correctness/Optimality. We now provide formal arguments that our approach for chains yields a correct assignment of β_i values for all tasks (Theorem 1) and is optimal in the sense that if the approach returns THE SYSTEM IS NOT SCHEDULABLE, then there is no assignment of β_is that would cause the system to become schedulable (Theorem 2).

Before we prove the two main theorems of the section, we prove a useful invariant for the *for-loop* in Line 5 of Algorithm 1. The remainder of this section assumes the fixed-preemption model where a preemption is always taken at a fixed-preemption point (i.e., incurring the teardown/startup costs). In this model, it can be shown that Equation 9 remains a necessary and sufficient condition for limited-preemption EDF schedulability. However, under other preemption models where we may skip/delay preemptions, Equation 9 is only a sufficient condition. Thus, only the correctness theorem will hold, and we leave an investigation of optimality for these more dynamic settings to future research.

▶ **Lemma 1.** *At the beginning of each iteration of the* for-loop *at Line 5 with t_d being the current testing set interval considered, the following statements hold:*

1. *For any $t < t_d$, Equation 9 is satisfied for the current set of β_i values.*
2. *For any $\tau_i \in \Gamma$ such that $D_i \leq t_d$, the value of β_i set by the algorithm is maximum (over all possible schedulable chunk-size configurations of Γ).*

Proof: Lemma 1 is, as expected, proved by induction.

Initialization: Initially t_d equals $D_{\min}$; the minimum deadline is the first non-zero value of the DBF function for all tasks. Thus, Statement 1 is true since at all prior timepoints the first term of Equation 9 is zero and Equation 9 reduces to $L \leq L$ per the arguments in Section 3.2. Statement 2 is also vacuously true since β_1 is set to its largest possible value $\max_{1 \leq j \leq n_1} (c(v_{1,j}) + q(v_{1,j}))$ by the previous loop and does not affect schedulability as τ_1 cannot block any other task (by nature of having the smallest relative deadline).

Maintenance: Let us consider the current testing-set point t_d. Let t_c be the testing-set point considered in the previous iteration of the *for-loop.* Assume that Statements 1 and 2 were true at the beginning of the *for-loop* for point t_c; we will show that the statements will hold for t_d.

For Statement 1, we must show that Equation 9 is not invalidated when we execute the *for-loop* for t_c. As was previously argued, for $t < t_c$, the DBF values are unchanged as any changes made to β_i values in the iteration for t_c only affect the $\widehat{C_i}$ of tasks with $D_i > t_c$. Thus, Statement 1 continues to hold for all $t < t_c$ by assumption. We only need to show that the previous iteration will set the corresponding β values such that Equation 9 will also be true at t_c. However, this obviously holds since for each τ_i with $D_i > t_c$, either β_i already satisfied Equation 9 (and does not change) or it is set by Line 11 of Algorithm 1 to the largest value that satisfies Equation 9 for the current values of β.

Statement 2 follows from the last observation of the previous paragraph and by the assumption that β values for all τ_i with $D_i \leq t_c$ are set to their maximum value by assumption and cannot change at t_c or after. Therefore, the $\widehat{C_i}$ values that contribute to the DBF in Equation 9 at t_c are as small as possible. It therefore follows that any τ_j with $t_c \leq D_j \leq t_d$ has either already had it's β_j value set to the largest possible to satisfy Equation 9 for some $t < t_c$ or has its value set to the largest possible to satisfy Equation 9 at t_c.

Termination: Let t_d be the last testing set interval considered by the for loop in Line 5. During the execution of the loop, the algorithm may return THE SYSTEM IS NOT SCHEDULABLE. In the case that not schedulable is returned, Statements 1 and 2 hold for all testing set intervals up to (and including) t_d, but not necessarily after t_d. If the algorithm completes execution of the loop without returning, the Statements 1 and 2 are guaranteed to hold for all testing set intervals in $\mathcal{T}(\widehat{\Gamma})_1$ (by properties of testing-sets for Equation 9 and that the last testing set point must be at least D_N). ☐

▶ **Lemma 2.** *In an implicit-deadline task system, Equation 9 will be satisfied at all points* $L > D_{\max}$ *if* $U \leq 1$.

Proof: By contradiction. Consider some $t_d > D_{\max}$ in the testing set at which Equation 9 fails to hold. At this point, there are no tasks where $D_i > t_d$, and so Equation 9 becomes $\sum_{\tau_i \in \Gamma} \text{DBF}_i(t_d) \leq t_d$. From the definition of the DBF:

$$\sum_{\tau_i \in \Gamma} \max\left(\left\lfloor \frac{t_d - D_i}{T_i} \right\rfloor + 1, 0\right) \cdot C_i > t_d$$

Since for an implicit-deadline task system, $D_i = T_i$ for all tasks τ_i,

$$\sum_{\tau_i \in \Gamma} \max\left(\left\lfloor \frac{t_d}{T_i} \right\rfloor, 0\right) \cdot C_i > t_d$$

From which it follows that

$$\sum_{\tau_i \in \Gamma} \max\left(\frac{t_d}{T_i}, 0\right) \cdot C_i > t_d$$

As t_d and T_i are both positive:

$$\sum_{\tau_i \in \Gamma} \frac{t_d}{T_i} \cdot C_i > t_d$$

which implies that $U > 1$. ☐

▶ **Theorem 1.** *If the schedulability test returns* THE SYSTEM IS SCHEDULABLE, *then assignment of* β_i *values by the algorithm in Algorithm 1 ensures that the task system meets all deadlines when scheduled by limited-preemption EDF.*

Proof: The termination argument of Lemma 1 argues that Statement 1 and therefore Equation 9 hold for all testing set points in $\mathcal{T}(\widehat{\Gamma})_1$. If the system is an implicit-deadline system, then by the check in line 17, the system utilization must be ≤ 1; by Lemma 2, Statement 1 will continue to hold for all points in $\mathcal{T}(\widehat{\Gamma})_2$. If the system is a constrained-deadline system, the loop in Line 21 will complete and return THE SYSTEM IS SCHEDULABLE if and only if Equation 9 holds for all points in $\mathcal{T}(\widehat{\Gamma})_2$.

Since Equation 9 is sufficient (and necessary for the fixed-preemption model), the task system Γ is schedulable by limited-preemption EDF when the assigned chunk-sizes β_i are used. ☐

▶ **Theorem 2.** *If the schedulability test returns* THE SYSTEM IS NOT SCHEDULABLE, *then there does not exist an assignment of β_i values such that Equation 6 is satisfied.*

Proof: If THE SYSTEM IS NOT SCHEDULABLE is returned while processing a testing-set interval t_d in $\mathcal{T}(\widehat{\Gamma})_1$ then either $\Delta(t_d) < 0$ or $\beta_i < q(v_{i,j})$ for some i and j. In either case, Statement 2 of Lemma 1 implies that the β_i's set prior to t_d are as large as possible with respect to $t < t_d$ for Equation 9. Therefore, if $\Delta(t_d) < 0$ is true, it is not possible to find another assignment of β_i's to make this false. Otherwise, if $\beta_i < q(v_{i,j})$, the fact that β_i was set to its maximum value means there does not exist a larger possible β_i to successfully fit task execution into given the startup/teardown costs $q(v_{i,j})$ of the phase $v_{i,j}$.

If THE SYSTEM IS NOT SCHEDULABLE is returned while processing a testing-set interval t_d in $\mathcal{T}(\widehat{\Gamma})_2$, then Equation 9 is false for some $t_d \in \mathcal{T}(\widehat{\Gamma})_2$. The right-side term of equation 9 considers only tasks where $D_i > L$; as $\mathcal{T}(\widehat{\Gamma})_2$ begins past $D_{\max}$, there can be no tasks matching this condition and therefore no assignment of β_i that would reduce the right-side term. Per Statement 2 of Lemma 1, each β_i has already been maximized when considering $\mathcal{T}(\widehat{\Gamma})_1$; reducing some β_i can only increase the number of preemption points required and therefore increase the DBF of some task in the left-side term of Equation 9. Therefore, there is no alternative assignment of β's that would reduce the left side of Equation 9 and cause the system to become schedulable. ☐

5 Systems of Conditional MPS Sporadic Tasks

In Section 4 we considered recurrent tasks representing 'linear' workflows: each task models a piece of straight-line code comprising a sequence of phases that are to be executed in order. In many event-driven real-time application systems, however, the code modeled by a task may include conditional constructs ("if-then-else" statements) in which the outcome of evaluating a condition depends upon factors (such as the current state of the system, the values of certain external variables, etc.), which only become known at run-time, and indeed may differ upon different invocations of the task. Hence the precise sequence of phases that is to be executed when a task is invoked is not known a priori. It is convenient to model such tasks as directed acyclic graphs (DAGs) in which the vertices represent execution of straight-line code, and a vertex representing a piece of straight-line code ending in a conditional expression has out-degree > 1 – see Figure 1 for an example. In this figure the vertex a denotes a piece of straight-line code that ends with the execution of a conditional expression. Depending upon the outcome of this execution, the code represented by either the vertex b or the vertex c executes, after which the code represented by the vertex d is executed. In this section, we briefly explain how our proposed Multi-Phase Secure (MPS) workload model may be further extended (i.e., beyond the aspects discussed in Section 4) to accommodate recurrent tasks that may include such conditional constructs.

5.1 Model

We now provide a more formal description of the *conditional MPS sporadic task model.* Each task τ_i is characterized by a 3-tuple (G_i, D_i, T_i) where D_i and T_i are the relative deadline and period, and G_i is a DAG: $G_i = (V_i, E_i)$ with $E_i \subsetneq V_i \times V_i$. Each vertex $v_{i,j} \in V_i$ represents a phase of

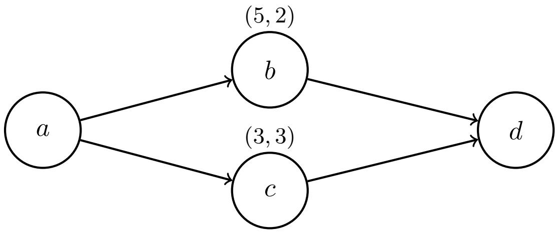


Figure 1 Each vertex characterized by a pair of values; the former one representing the WCET of the node/phase, $c(v)$ and the latter one representing the sum of the startup and teardown cost, $q(v)$. We assume that the code represented by vertices a and d execute using the same security mechanism, whereas the code represented by vertices b and c each execute using a distinct different mechanism.

computation, which must execute using a specified security mechanism. The interpretation of each edge $(v_{i,j}, v_{i,k}) \in E$ depends upon the outdegree (i.e., the number of outgoing edges) of vertex $v_{i,j}$:

1. If this outdegree $= 1$, then the edge denotes a precedence constraint: vertex $v_{i,k}$ may only begin to execute after vertex $v_{i,j}$ has completed execution.
2. If this outdegree is ≥ 2, then all the outgoing edges from $v_{i,j}$ collectively denote the choices available upon the execution of a conditional construct: after $v_{i,j}$ completes execution, exactly one of the vertices $v_{i,k}$ for which $(v_{i,j}, v_{i,k}) \in E$ becomes eligible to start executing. It is not known beforehand which one this may be, and different ones may become eligible upon different invocations of the task.

A WCET function $c : V_i \to \mathbb{N}$ is specified, with $c(v_{i,j})$ denoting the WCET of node $v_{i,j} \in V_i$. An overhead function $q : V_i \to \mathbb{N}$ is specified, with $q(v_{i,j})$ denoting the startup/teardown cost associated with the security mechanism using which vertex $v_{i,j}$ is to execute.

A Simplifying Assumption. In this paper, we assume that during the execution of an invocation of task τ_i the teardown cost associated with the security mechanism of $v_{i,j}$, and the startup cost associated with the security mechanism of $v_{i,k}$, is always paid upon traversing an edge $(v_{i,j}, v_{i,k})$. This will indeed be the case if the indegree of $v_{i,j}$ is equal to 1 – in our model, there is no reason to split out a piece of straight-line code that executes using the same security mechanism into two separate nodes. However, for the case that the indegree of $v_{i,j}$ is greater than 1 it is possible that some of the predecessor vertices of $v_{i,j}$, which represent pieces of code that may conditionally lead to the execution of vertex $v_{i,j}$, are to execute using the same security mechanism as vertex $v_{i,j}$. Since both vertices execute using the same security mechanism upon traversing such an edge during run-time, there is no need to pay the teardown and start-up costs between them and our simplifying assumption represents a conservative over-approximation; we leave the task of eliminating such over-approximation to future work. (We point out here that getting rid of this simplifying assumption would also require us to move away from fixed-preemption point model [28] — the assumption, stated in Section 4, that preemption points are statically assigned prior to run-time.)

A Subtlety. We now highlight an issue, not present during the consideration of linear workflows in Section 4, that arises when we are dealing with conditional code. As stated above, it is not known prior to run-time which of the branches through the conditional code will be taken during run-time (and this may differ upon different invocations). Since hard-real-time systems are required to meet their deadlines under all circumstances that may legally occur during run-time, during pre-runtime schedulability analysis one makes the conservative assumption that each invocation of the task takes the "longest" path —the one with maximum cumulative execution requirement— through the DAG.

Standard algorithms are known for identifying the longest path through a DAG that have running time linear in the representation of the DAG. Under limited-preemption scheduling, however, identifying the path through the DAG that has maximum cumulative execution requirement is not entirely straightforward. Let us consider again the example DAG of Figure 1.

- If the chunk size β_i for this task were ≥ 7, then both branches can be executed non-preemptively. The upper branch incurs a cost of $5 + 2 = \mathbf{7}$ while for the lower branch the cost is $3 + 3 = \mathbf{6}$; therefore, the upper branch is the computationally more expensive one.
- Now suppose $\beta_i = 4$. Then the upper branch may need to execute in $\lceil 5/(4-2) \rceil$ or 3 contiguous pieces, for a cumulative cost of $5 + 3 \times 2 = \mathbf{11}$. The lower branch may need to execute in $\lceil 3/(4-3) \rceil$ or 3 contiguous pieces, for a cumulative cost of $3 + 3 \times 3 = \mathbf{12}$, and is hence the more expensive branch.

This example illustrates that the computationally most expensive path through a DAG that represents conditional execution depends upon the value of the chunk size parameter (the β_i parameter of the task). And as we saw in Section 4, our approach to scheduling MPS sporadic tasks has been to convert each task to a limited-preemption sporadic task; our procedure for doing so (Algorithm 1) repeatedly changes the values of the β_i parameters of the tasks. As we attempt here to adapt the techniques of Section 4 to schedulability analysis of conditional MPS tasks, we must remain cognizant of this fact and account for it in the schedulability analysis algorithms that we will develop here.

5.2 Overview of Approach

Given task system Γ comprising multiple independent conditional MDS tasks to be scheduled upon a single preemptive processor, we will adopt an approach similar to the one described in Section 4:

- As in Section 4, we will first execute a schedulability analysis algorithm that constructs a limited-preemption task $\widehat{\tau_i}$ corresponding to each task $\tau_i \in \Gamma$, and determines whether the resulting limited-preemption task system can be scheduled by the limited-preemption EDF scheduling algorithm [4, 8] to always meet all deadlines.
- If so, then during run-time we will schedule the original task system using the limited-preemption EDF scheduling algorithm, with chunk-sizes as determined for the constructed limited-preemption tasks.

In defining the mapping from the given set of conditional MPS sporadic tasks to limited-preemption sporadic tasks, we must be cognizant of the issue identified above —that changing the value of β_i may change the cumulative worst-case execution time associated with τ_i— and adapt the algorithm of Section 4 accordingly; below we describe how we do so.

The Schedulability Test. The schedulability test, which is an adaptation, to deal with the issue identified above, of the one discussed in Section 4, is presented in pseudo-code form in Algorithm 2; we discuss it briefly below. As in Section 4, limited-preemption task $\widehat{\tau_i}$ is assigned the same relative deadline parameter value (i.e., D_i) and the same period parameter value (i.e., T_i) as τ_i; its WCET $\widehat{C_i}$ and its chunk-size parameter β_i are computed as described below.

We introduce integer variables $\text{cnt}(v_{i,j})$ for each i, $1 \leq i \leq n$, and for each j such that $v_{i,j} \in V_i$, to denote the maximum number of contiguous time-intervals in which the j'th phase of τ_i executes. Then $\widehat{C_i}$, the WCET of each job of task $\widehat{\tau_i}$, can be written as

$$\widehat{C_i} \stackrel{\text{def}}{=} \max_{(\text{all paths } \mathcal{P} \text{ in } G_i)} \sum_{v_{i,j} \in \mathcal{P}} \Big(c(v_{i,j}) + \text{cnt}(v_{i,j}) \times q(v_{i,j}) \Big) \tag{15}$$

Algorithm 2 The Preprocessing Algorithm for Systems of Conditional MPS Sporadic Tasks (see Section 5)

Input: (Γ)
for *each task* $\tau_i = \big(G_i = (V_i, E_i), D_i, T_i\big)$ *in* Γ **do**
  for *each vertex* $v_{i,j} \in V_i$ **do**
    $\text{cnt}(v_{i,j}) \leftarrow 1$
  $\beta_i \leftarrow \max_{v_{i,j} \in V_i} (c(v_{i,j}) + q(v_{i,j}))$
for t_d *iterating in increasing order over* $\mathcal{T}(\widehat{\Gamma})_1$ **do**
  Compute $\Delta(t_d)$ as per Eqn 13
  if $\Delta(t_d) < 0$ **then**
    return THE SYSTEM IS NOT SCHEDULABLE
  for *each task* $\tau_i = \big(G_i = (V_i, E_i), D_i, T_i\big)$ *for which* $D_i > t_d$ **do**
    if *($\beta_i > \Delta(t_d)$)* **then**
      $\beta_i \leftarrow \Delta(t_d)$
      for *each vertex* $v_{i,j} \in V_i$ **do**
        $\text{cnt}(v_{i,j}) \leftarrow \min_{\kappa \in \mathbb{N}} \text{such that} \frac{c(v_{i,j})}{\kappa} + q(v_{i,j}) \leq \beta_i$
if *utilization* > 1 **then**
  return THE SYSTEM IS NOT SCHEDULABLE
if *constrained-deadline system* **then**
  for t_d *iterating in increasing order over* $\mathcal{T}(\widehat{\Gamma})_2$ **do**
    Compute $\Delta(t_d)$ as per Eqn 13
    if $\Delta(t_d) < 0$ **then**
      return THE SYSTEM IS NOT SCHEDULABLE
return THE SYSTEM IS SCHEDULABLE

Notice the difference with Equation 11 in Section 4: since the "longest" path (i.e., the one with maximum cumulative execution requirement) may change as β_i changes, $\widehat{C_i}$ is computed as the maximum cumulative WCET over all paths in the DAG. (We point out that a change in β_i gets reflected in Equation 15 as a change in the values assigned to the $\text{cnt}(v_{i,j})$ variables.) As previously stated, this computation takes time linear in the representation of the DAG.

As with linear tasks, once we reach D_{max} (the end of $\mathcal{T}(\Gamma)_1$), the β_i value of each task, and therefore the value of $\widehat{C_i}$ is fixed. Therefore, the claim from Lemma 2 continues to hold, and for implicit-deadline task systems, we can continue to perform a simple utilization check on the system rather than testing the remainder of the testing set.

6 Empirical Evaluation

In the previous sections, we have developed algorithms to introduce preemptions into the execution of the phases of multi-phase secure tasks. While these preemptions reduce the blocking that higher-priority phases/jobs experience at runtime, they come at a cost – increased overhead due to the additional teardown/startup costs that must be performed before and after every inserted preemption. Thus, while the limited-preemption approach proposed in this paper theoretically dominates the non-preemptive approach (i.e., each phase is executed non-preemptively and preemptions are permitted only between phases of a task), it is unclear how much schedulability improvement *on average* we can expect a system designer to obtain from using our proposed approach. In this section, we provide an initial empirical analysis on that topic via the application of the schedulability tests of Section 4 (over chains respectively) over synthetically-generated tasks systems. We compare the proposed schedulability tests with existing limited-preemption scheduling where each phase of a task is executed fully non-preemptively.

In this work, we limit our scope to evaluating linear task sequences; while we believe our algorithm should achieve similar results for schedulability improvements and performance on a DAG-based conditional task model, we defer further evaluation, analysis, and refinement of the conditional model to future work.

6.1 Experimental Setup

The evaluation was conducted using a C++ simulation. All tests were performed on a a server with two Intel Xeon Gold 6130 (Skylake) processors running at 2.1 GHz, and with 64GB of memory. Multiple task sets were evaluated in parallel; each task set was given a single thread on which to run. We evaluate task sets for many parameter variations, including task count, number of phases per task, and utilization. For each combination of parameters, we generate 1000 random task systems using the UUniFast algorithm [9], first to assign a total utilization $\frac{C_i}{T_i}$ to each task and then to distribute the task's total allotted time C_i between the execution times $c_{(v_{i,j})}$ and startup/teardown overhead $q_{(v_{i,j})}$ for each of its phases. We note that compared to the prior work in [5] that this paper extends, we have adjusted this distribution to be more consistent and to be independent of the number of phases in the task. We then evaluate each task set against three algorithms: *chains*, the algorithm presented in section 4, *phase NP*, in which there is a static preemption point between each phase but no additional preemption points can be inserted, and *fully NP*, in which the entire task runs as a single non-preemptive chunk.

We evaluate both implicit-deadline and constrained-deadline task systems. In an implicit-deadline system, $D_i = T_i$ for each task. In a constrained-deadline system, we choose a random value for each D_i that is uniformly distributed between the task's execution time and its period T_i.

For implicit-deadline systems, we evaluate U at increments of 0.1 in $[0, 1]$. We note that, compared to the prior version of this work in [5], the improvement to the testing set described in Section 4 allows us to evaluate these systems in a reasonable amount of time even for large numbers of tasks; to understand the impact of this optimization, we also test the exponential set presented in the original version on systems with 8 tasks or fewer and compare their execution times. For constrained-deadline systems, the testing set is bounded by a term that is determined in part by a factor $\frac{1}{1-U}$; as $U \to 1$, the testing set size becomes very large and becomes infeasible to evaluate. We therefore limit our evaluation of constrained-deadline systems to utilizations in $[0, 0.9]$.

6.2 Simulation Results

Schedulability of Implicit-Deadline Systems. Figure 2 shows the schedulability ratio of each of the three tests described above — chains, phase NP, and fully NP. For each test, tasks with low utilization are all schedulable, but schedulability decreases as the utilization of the system increases. By inserting additional preemption points, the chains algorithm is able to obtain a schedulability improvement over a phase NP approach on these higher-utilization tasks. Once utilization reaches 1, it is not possible to insert any preemption points, as inserting a preemption

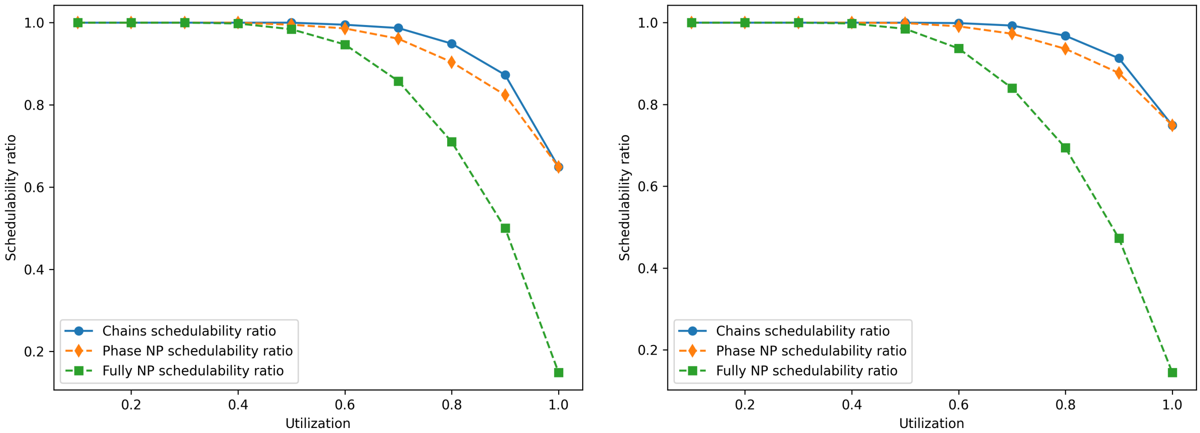


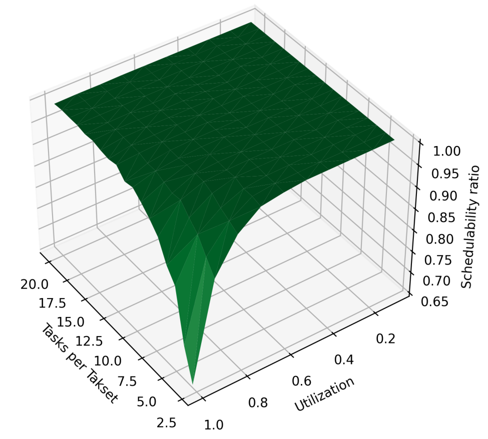
Figure 2 Schedulability ratio of implicit-deadline tasksets with 3 tasks each and 10–30ms periods. On the left, tasks are randomly assigned 1–4 phases; on the right, 1–6.

point would increase the startup/teardown overhead and cause the utilization to exceed 1; therefore, the chains algorithm does not lead to a schedulability improvement. Tasks with 1-6 phases have a slightly higher schedulability ratio than tasks with 1-4 phases; we explore this effect more in the paragraph below.

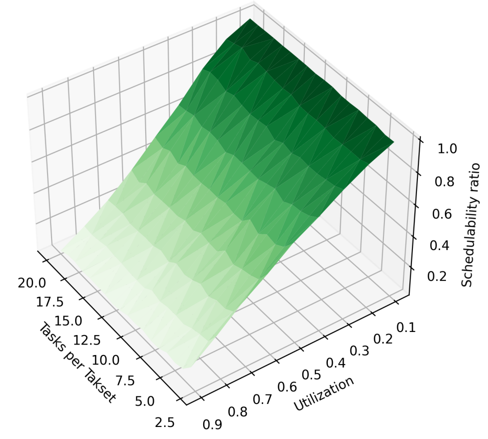
Figure 3 shows the impact of various parameters of the task system on the schedulability ratio using the chains algorithm and 10-30ms periods. In Figure 3a, tasks are randomly assigned 1-4 phases, while the utilization and tasks per taskset are varied. In Figure 3c, the number of tasks is held constant at 3, and each taskset is assigned a fixed number of phases, varying from 2 to 20 phases per task. In Figure 3e, the utilization is held constant at 0.9. As in Figure 2, increasing utilization reduces the schedulability ratio. Here, it is clearly visible that increasing the number of tasks or number of phases per task improves the schedulability ratio. When these parameters are increased, the system's total execution time is divided among more tasks or among more phases, so that each task has a lower blocking time. The reduction in blocking time makes the system more likely to be schedulable.

For completeness, we also evaluate the performance of the algorithm on a wider 1–1000ms period range, using both a uniform distribution of periods within this range, as well as the log-uniform distribution recommended in [16]. Figure 4 shows how the schedulability of these task sets compares to the tasksets with 10-30ms periods. The schedulability of the sets with the wider period range is much lower; in particular, tasksets containing a task with a small period are less likely to be schedulable. We hypothesize that this is due to these high-frequency tasks having less ability to expand their execution time as preemption points are inserted. This trend is also apparent in Figure 5; unlike tasks with 10–30ms periods, increasing the number of tasks increases the probability of generating a high-frequency task and therefore causes the schedulability ratio to decrease.

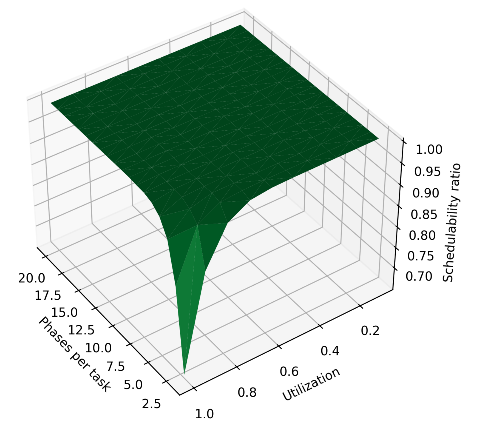
Schedulability of Constrained-Deadline Systems. Figure 6 shows the schedulability ratio for each of the three scheduling algorithms on constrained-deadline tasksets, in which all other task parameters follow the same configuration as Figure 2. In a constrained-deadline system, the chains algorithm is also able to obtain a schedulability improvement over a phase NP approach, by inserting additional preemption points to reduce the blocking time. As the tasks in these systems have shorter deadlines than the implicit-deadline systems, all three algorithms obtain a lower schedulability ratio.



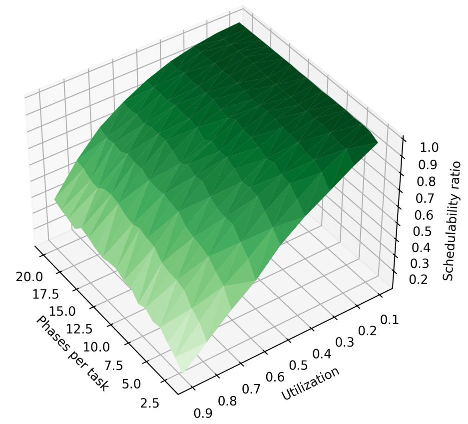
(a) Implicit deadlines, 1–4 phases.



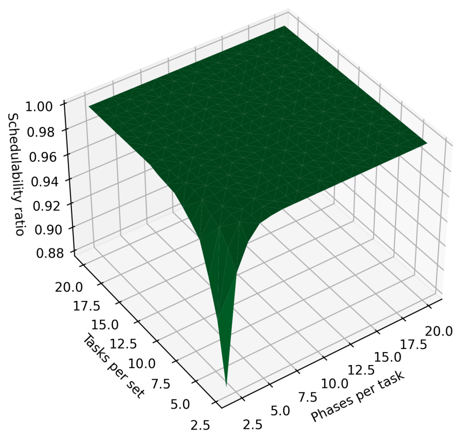
(b) Constrained deadlines, 1–4 phases.



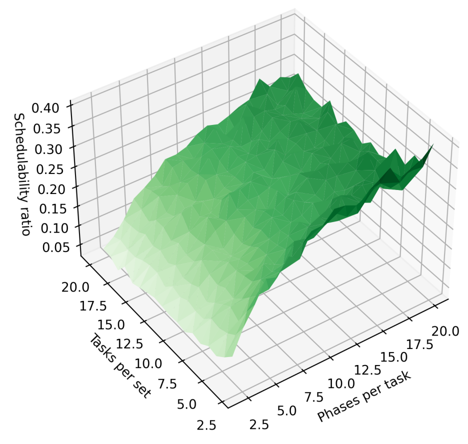
(c) Implicit deadlines, 3 tasks per set.



(d) Constrained deadlines, 3 tasks per set.



(e) Implicit deadlines, utilization 0.9.



(f) Constrained deadlines, utilization 0.9.

Figure 3 Schedulability ratio of chains algorithm when varying taskset parameters. All tasks have periods selected uniformly from 10–30ms.

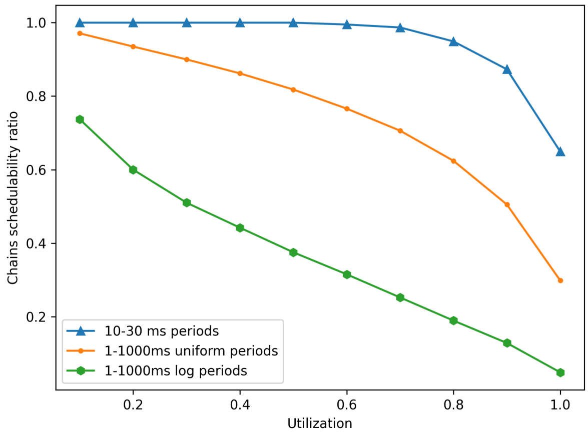


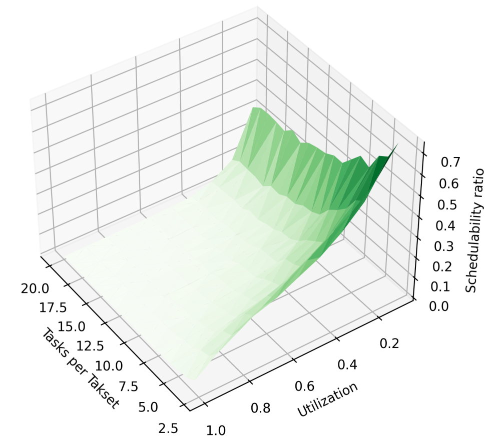
Figure 4 Comparison of the effect on schedulability of changing the period range of the generated tasks. Each set contains 3 implicit-deadline tasks with 1–4 phases.

The right column of Figure 3 shows the effect of varying different taskset parameters for the constrained-deadline task sets, which otherwise have the same parameter configurations as the implicit-deadline systems. For constrained-deadline tasks, the schedulability ratio is generally lower, and the effect of increasing the phase count is smaller. Increasing the number of tasks has only a very small effect on the schedulability ratio. We hypothesize that, although increasing the task count still tends to reduce the system's blocking times, it also increases the probability that one or more tasks will have a tightly-constrained deadline, which reduces the chance that the system will be schedulable.

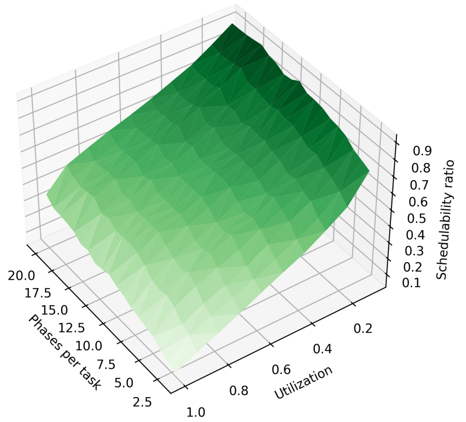
Performance. In our prior work [5], tasks with 6 or more tasks took over 24 hours to run, and were therefore infeasible to evaluate. In this work, we rewrite the original Python-based simulation using C++, and correct an implementation issue with the original construction of the hyperperiod-bounded testing set. These changes lead to significant performance improvements on their own.

In this work, we also introduce an optimization for implicit-deadline task systems that allows us to avoid testing timepoints past $D_{\max}$. In Figure 7, we analyze the impact of this optimization. Using the full, exponential testing set from the prior work, our implementation is able to test tasksets with 6 tasks in only a few seconds, but the time needed still scales exponentially with the number of tasks; past eight tasks per set, the analysis once again takes an unreasonable amount of time to run. Using the optimization for implicit-deadline systems, evaluating schedulability is orders of magnitude faster, and the evaluation times also become more consistent for each taskset. This optimization allows us to determine the schedulability ratio for systems of up to 20 tasks, the results of which are displayed in Figure 3. As the time needed to determine schedulability now scales much more slowly with respect to the number of tasks, we believe that evaluating even larger task sets is also possible.

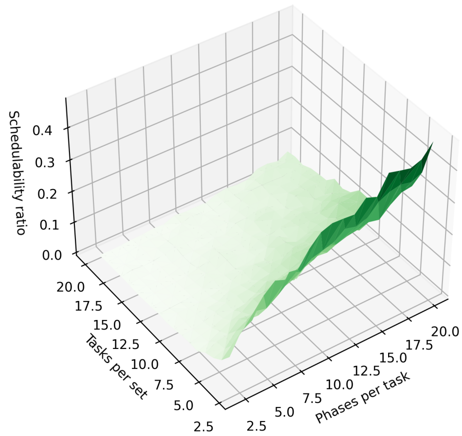
In Figure 8, we evaluate the performance of determining schedulability of constrained-deadline task sets using the pseudo-polynomial bound on the testing set identified in [3]. Compared to the full hyperperiod testing set, running the evaluation with this testing set is much faster. However, we note that it is still slower than the optimized implicit-deadline bound from Figure 7. Moreover, the distribution of execution times exhibits high variability. Perhaps most importantly, our evaluation is on task sets where $U = 0.9$, at which the pseudo-polynomial bound is still reasonably small. As shown in Figure 9, when $U \to 1$, the size of the pseudo-polynomial set approaches the



(a) Implicit deadlines, 1–4 phases.



(b) Implicit deadlines, 3 tasks per set.



(c) Implicit deadlines, utilization 0.9.

Figure 5 Schedulability ratio of chains algorithm when varying taskset parameters. All tasks have periods selected from 1–1000ms using a log-uniform distribution.

hyperperiod set, and it becomes increasingly less feasible to iterate over the entire testing set.

7 Conclusions

We believe that the concurrent consideration of timing and security properties within a single unified framework is an effective means of extending the rigorous approach of real-time scheduling theory to guaranteeing appropriately-articulated security properties in resource-constrained embedded systems. In real-time scheduling theory, pre run-time verification of timing correctness is performed using models of run-time behavior; these models are carefully crafted for specific purposes: e..g, the sporadic task model [3] has been designed to represent recurrent processes for which it is safe to assume a minimum duration between successive invocations and for which timing correctness is defined as the ability to meet all deadlines.

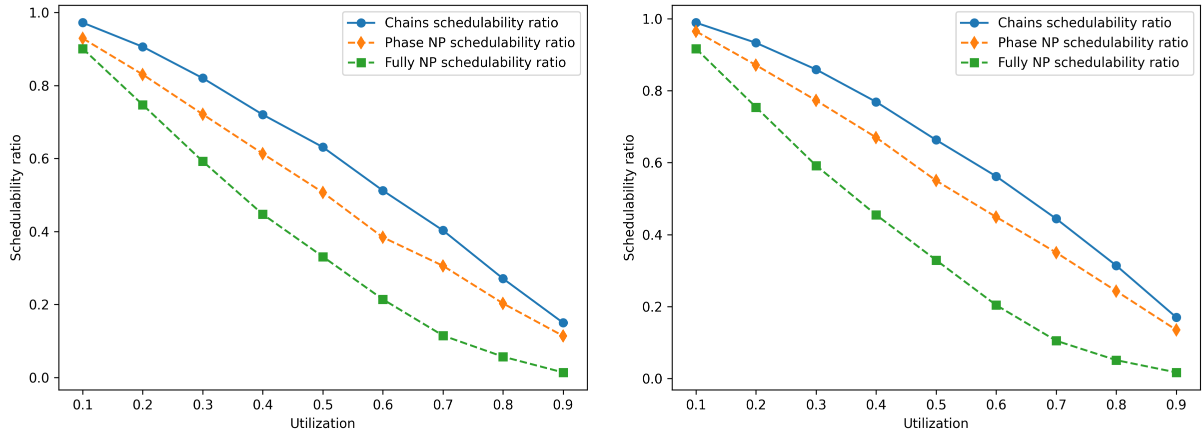


Figure 6 Comparison of schedulability ratios. Each set has 3 constrained-deadline tasks with periods selected uniformly from 10–30ms. On the left, tasks are randomly assigned 1–4 phases; on the right, 1–6.

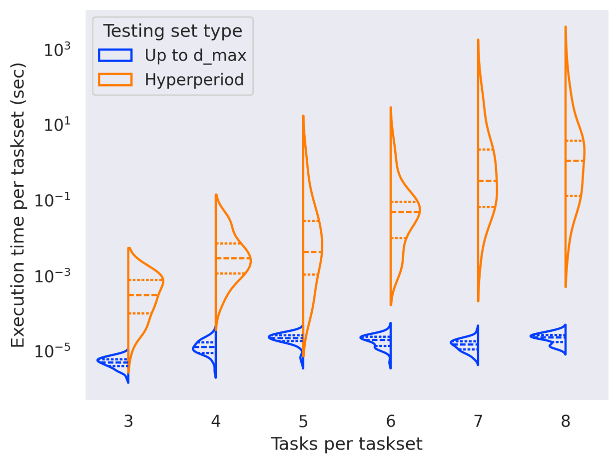


Figure 7 Time to test schedulability of sets of 3 implicit-deadline tasks with 10–30ms periods, 1–4 phases, and utilization of 0.9, either testing up to the hyperperiod or stopping at $D_{\max}$. Note the logarithmic scale.

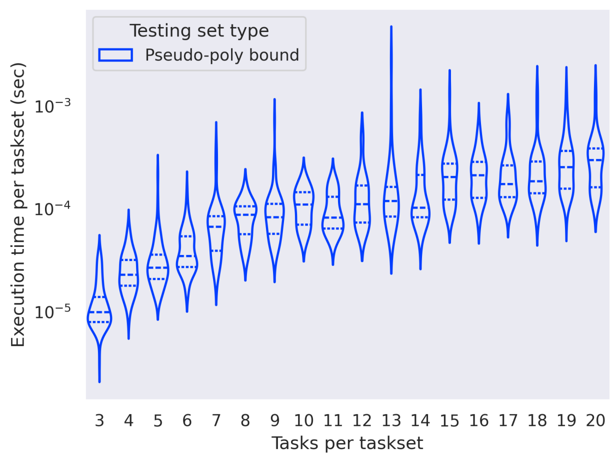


Figure 8 Time to test schedulability of sets of 3 constrained-deadline tasks, with 10–30ms periods, 1–4 phases, and utilization of 0.9, using the psuedo-polynomial testing set bound from [3]. Note the logarithmic scale.

Our algorithm, which we first introduced in [5], extends the sporadic task model, and a generalization that allows for the modeling of conditional code, in a security-cognizant manner, to deal with a particular kind of security model. For the specific model that we have proposed, we have developed algorithms that are able to provide provable correctness of both the timing and the security properties that are considered.

In this extension to our original work, we have reduced the execution time complexity of our algorithm, and demonstrated that in an implicit-deadline task system, it can run quickly even for large numbers of tasks. In combination with a more efficient implementation of the algorithm, this allowed us to evaluate larger and more complex task systems. We have also illustrated the scaling properties of the algorithm's execution time for both constrained- and implicit-deadline systems, and shown how its ability to schedule tasks is affected by varying their properties. Finally, we have clarified several details of our algorithm and have provided a more complete demonstration of the

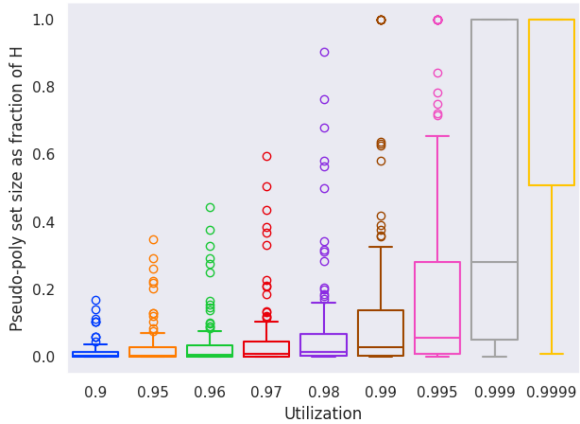


Figure 9 Comparison of the sizes of the hyperperiod and pseudo-polynomial testing sets. For each utilization, we generated 100 sets of 5 constrained-deadline tasks with periods from 10–30, and 1–4 phases.

improved schedulability offered by our algorithm over a phase or fully non-preemptive approach.

Although the work described in this manuscript arose out of our related projects in embedded systems security, we emphasize that we are not claiming that our algorithms solve any security problems; rather, they solve a scheduling problem that may arise from a class of security problems for which an adequate protective response gives rise to execution environments with bounded-cost startup/teardown operations. As future work, we intend to evaluate these scheduling models in conjunction with real-world attack/defense models.

On the other hand, we believe that our results are relevant beyond just security considerations: that they may, in fact, be considered to be further contributions to the real-time scheduling theory literature dealing with limited-preemption scheduling. They may also be extended to other limited-preemption scheduling frameworks, e.g., for multi-core platforms.

References

1. N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems*, 8:173–198, 1995.
2. S. Baruah, R. Howell, and L. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324, 1990.
3. S. Baruah, A. Mok, and L. Rosier. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida, 1990. IEEE Computer Society Press.
4. Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, 2005.
5. Sanjoy Baruah, Thidapat Chantem, Nathan Fisher, and Fatima Raadia. A scheduling model inspired by security considerations. In *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 32–41, 2023. `doi:10.1109/ISORC58943.2023.00016`.
6. Sanjoy K. Baruah. Security-cognizant real-time scheduling. In *25th IEEE International Symposium On Real-Time Distributed Computing, ISORC 2022, Västerås, Sweden, May 17-18, 2022*, pages 1–9. IEEE, 2022. `doi:10.1109/ISORC52572.2022.9812766`.
7. Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.*, 2(4):301–324, November 1990.
8. Marko Bertogna and Sanjoy Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 2010.
9. Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), 2005.
10. Giorgio C Buttazzo, Marko Bertogna, and Gang Yao. Limited preemptive scheduling for real-time systems. a survey. *IEEE Trans. Industr. Inform.*, 9(1):3–15, February 2013.
11. John Cavicchio and Nathan Fisher. Integrating preemption thresholds with limited preemption

scheduling. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, August 2020.

12 Friedrich Eisenbrand and Thomas Rothvoß. EDF-schedulability of synchronous periodic task systems is coNP-hard. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2010.

13 P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks remains coNP-complete under bounded utilization. In *2015 IEEE Real-Time Systems Symposium*, pages 87–95, 2015.

14 P. Ekberg and W. Yi. Uniprocessor feasibility of sporadic tasks with constrained deadlines is strongly coNP-complete. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 281–286, 2015.

15 Pontus Ekberg. *Models and Complexity Results in Real-Time Scheduling Theory*. PhD thesis, Ph.D. thesis, Uppsala University, 2015.

16 P. Emberson, R. Stafford, and R.I. Davis. Techniques for the synthesis of multiprocessor tasksets. In *WATERS workshop at the Euromicro Conference on Real-Time Systems*, pages 6–11, July 2010. 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems ; Conference date: 06-07-2010.

17 K. Jiang, A. Lifa, P. Eles, Z. Peng, and W. Jiang. Energy-aware design of secure multi-mode real-time embedded systems with FPGA co-processors. In *Proc. Int. Conf. Real-Time Networks and Systems*, pages 109–118, October 2013.

18 M. Lin, L. Xu, L.T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu. Static security optimization for real-time systems. *IEEE Trans. Industrial Informatics*, 5(1):22–37, February 2009.

19 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

20 Y. Ma, W. Jiang, N. Sang, and X. Zhang. ARCSM: A distributed feedback control mechanism for security-critical real-time system. In *Proc. Int. Symp. Parallel and Distributed Processing with Applications*, pages 379–386, July 2012.

21 Tanmaya Mishra, Thidapat Chantem, and Ryan Gerdes. Teecheck: Securing intra-vehicular communication using trusted execution. In *Proceedings of the 28th International Conference on Real-Time Networks and Systems*, RTNS 2020, page 128–138, New York, NY, USA, 2020. Association for Computing Machinery.

22 Sibin Mohan, Man Ki Yoon, Rodolfo Pellizzoni, and Rakesh Bobba. Real-time systems security through scheduler constraints. In *Proceedings of the 2014 Agile Conference*, AGILE '14, pages 129–140, 2014.

23 Anway Mukherjee, Tanmaya Mishra, Thidapat Chantem, Nathan Fisher, and Ryan Gerdes. Optimized trusted execution for hard real-time applications on cots processors. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, RTNS '19, page 50?60, New York, NY, USA, 2019. Association for Computing Machinery. `doi:10.1145/3356401.3356419`.

24 M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes. On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 103–116, April 2019. `doi:10.1109/RTAS.2019.00017`.

25 Mitra Nasri, Geoffrey Nelissen, and Gerhard Fohler. A new approach for limited preemptive scheduling in systems with preemption overhead. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*. IEEE, July 2016.

26 M. Pajic, N. Bezzo, J. Weimer, R. Alur, R. Mangharam, N. Michael, G.J. Pappas, O. Sokolsky, P. Tabuada, S. Weirich, and I. Lee. Towards synthesis of platform-aware attack-resilient control systems. In *Proc. Int. Conf. High Confidence Networked Systems*, pages 75–76, April 2013.

27 T. Xie and X. Qin. Scheduling security-critical real-time applications on clusters. *IEEE Trans. Computers*, 55(7):864–879, July 2006.

28 Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Feasibility analysis under fixed priority scheduling with fixed preemption points. In *2010 IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 71–80, 2010. `doi:10.1109/RTCSA.2010.40`.

29 M. Yoon, S. Mohan, C. Chen, and L. Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12, April 2016.

30 Man-Ki Yoon, Sibin Mohan, Chien-Ying Chen, and Lui Sha. Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–12. IEEE, 2016.