

# Limited-Preemption EDF Scheduling for Multi-Phase Secure Tasks

Benjamin Standaert  

Washington University in St. Louis, United States

Fatima Raadia  

Wayne State University, United States

Marion Sudvarg<sup>1</sup>   

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 execu-

tion using different security mechanisms which each have associated execution time costs for startup and teardown. We develop corresponding limited-preemption EDF scheduling algorithms and associated pseudo-polynomial schedulability tests for constrained-deadline MPS tasks. In doing so, we provide a correction to a long-standing schedulability condition for EDF under limited-preemption. Evaluation shows that the proposed tests 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

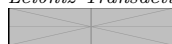
<sup>1</sup> Corresponding author



© Benjamin Standaert, Fatima Raadia, Marion Sudvarg, Sanjoy Baruah, Thidapat Chantem, Nathan Fisher, Christopher Gill;

licensed under Creative Commons License CC-BY 4.0

*Leibniz Transactions on Embedded Systems*, Vol. 1, Issue 1, Article No. 42, pp. 42:1–42:27



Leibniz Transactions on Embedded Systems

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## 1 Introduction

In today’s interconnected world, the security of real-time systems has emerged as a primary concern, e.g., [42, 26, 29, 23, 38, 45, 34], given the widespread integration of electronic devices into 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 ensure that another task cannot deduce when an event of interest occurs [34].

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. 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 incurred by, e.g., hardware mode switching, saving and restoring the stack, and copying data, resulting in 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 [35].

A recent ISORC paper [10] 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 [36] 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 [3] and the randomization-based schedule obfuscation approaches, e.g., the one in [46]) to such threats.

**Security-Cognizant Scheduling.** We believe the methodology formalized and used in [36] holds great promise as a means of integrating security and timing correctness concerns within a common framework. This methodology was articulated in [10] 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 [8], which applied the methodology articulated in [10] to the following problem in system design for real-time + security. We consider

64 computer platforms upon which *multiple different security mechanisms* (such as TEEs, encryption/decryption co-processors, FPGA-implemented secure computations, etc.) co-exist. Depending  
65 upon their security requirements, different pieces/parts of the (real-time) code may need to use  
66 different security mechanisms at different times. We therefore assume that the code is broken up  
67 into *phases*, with different consecutive phases needing to use different sets of security mechanisms  
68 – the security mechanisms used by each phase are specified for the phase. We assume that there is  
69 a startup/teardown overhead cost (for data communication, initialization, etc.) expressed as an  
70 execution duration, associated with switching between different security mechanisms. In other  
71 words, *there is a time overhead associated with switching between the execution of different phases*.  
72

73 **Contributions.** As in our prior work that this paper extends [8], we formalize the workload  
74 model discussed above as the *Multi-Phase Secure (MPS)* task model, with multiple independent  
75 recurrent processes of this kind that are to execute upon a single shared preemptive processor. We  
76 start out in Section 4 assuming that each recurrent process is represented using the widely-used  
77 3-parameter sporadic task model [6]. For this model, we represent the problem of ensuring  
78 timeliness plus security as a schedulability analysis problem, which we then solve by adapting  
79 results obtained in prior work (e.g., [7, 12, 16, 37, 17]) on limited-preemption scheduling. Later in  
80 Section 5, we propose a generalization that models conditional execution within each recurrent  
81 process. Its original treatment in [8] makes a simplifying assumption that was later addressed in  
82 another extension; we therefore present the conditional model and the simplifying assumption in  
83 this paper, but leave the analysis for conditional execution to the more accurate treatment in [39].  
84

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

93 **Extensions to the Prior Work.** This paper **extends, corrects, and clarifies** our prior results  
94 in [8], making the following new contributions. Section 3.2 presents a correction to the condition  
95 in [7, 12] for EDF schedulability of limited-preemption tasks. Section 4 leverages the corrected  
96 condition to introduce pseudo-polynomial schedulability analysis for sets of MPS tasks, improving  
97 the execution time of the associated algorithms originally proposed in [8], especially for tasks with  
98 implicit deadlines. With these improvements, we are able to evaluate larger sets of tasks with  
99 more realistic ranges of periods. We also address inconsistencies in [8] between the theoretical  
100 model and its implementation by using a continuous-time representation of the algorithms. These  
101 are reflected in the new results in Section 6, which more clearly demonstrate the advantages of  
102 our approach over non-preemptive schedulers.

103 **Organization.** The remainder of this manuscript is organized as follows. After briefly discussing  
104 some related scheduling-theory results in Section 2, Section 3.2 presents the correction to the  
105 limited-preemption EDF schedulability condition of [7, 12]. We then motivate and formally define  
106 the MPS sporadic tasks model in Section 4, and provide both pre-runtime analysis and a run-time  
107 scheduling algorithm for MPS sporadic task systems upon preemptive uniprocessor platforms. In  
108 Section 5 we further generalize the workload model to be able to represent conditional execution.  
109 We have performed schedulability experiments to evaluate both the effectiveness of our algorithms  
110 and the performance improvements over their original implementations in [8]. We report the  
111 results in Section 6. We conclude in Section 8 by pointing out some directions in which we

111 intend to extend this work, and by placing our results within a larger context on the timing- and  
112 security-aware synthesis of safety-critical systems.

## 113 **2 Some Real-Time Scheduling Background**

### 114 **2.1 The Sporadic Task Model [6]**

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

123 **Processor Demand Analysis (PDA).** A sporadic task system can be scheduled optimally by  
124 the Earliest Deadline First (EDF) [27] scheduling algorithm, given a preemptive uniprocessor. To  
125 determine whether a specific task system can be correctly scheduled by EDF, Processor Demand  
126 Analysis (PDA) [11] can be utilized. PDA is a necessary and sufficient algorithm that is also  
127 optimal. The key idea of PDA is built upon the *demand bound function* (DBF). Given an interval  
128 length of  $L$  such that  $L \geq 0$ , the DBF for a sporadic task  $\tau_i$  can be represented by  $\text{DBF}_i(L)$ : the  
129 maximum possible aggregate execution time required by jobs of task  $\tau_i$  such that they arrive in  $L$   
130 and have deadlines before  $L$ . The following equation was derived in [6] to compute its value:

$$131 \quad \text{DBF}_i(L) = \max \left( \left\lfloor \frac{L - D_i}{T_i} \right\rfloor + 1, 0 \right) \times C_i \quad (1)$$

132 For a task system  $\tau$  to be correctly scheduled by EDF, the following was derived in [6] as a  
133 necessary and sufficient condition for all  $L \geq 0$ :

$$134 \quad \left[ \left( \sum_{\tau_i \in \Gamma} \text{DBF}_i(L) \right) \leq L \right] \quad (2)$$

135 **The Testing Set.** A naïve application of PDA requires testing the validity of Equation 2 for all  
136 intervals. However, a more efficient approach, outlined in [6], involves checking only values of  $L$   
137 that follow the pattern  $L \equiv (k \times T_i + D_i)$  for some non-negative integer  $k$  and some  $\tau_i \in \Gamma$ .

138 Furthermore, it suffices to test such values that are less than the least common multiple of all  
139 the  $T_i$  parameters. The collection of all such values of  $t$  for which it is necessary to verify that  
140 Condition 2 holds true in order to confirm EDF-schedulability is referred to as the *testing set* for  
141 the sporadic task system  $\Gamma$ , often denoted as  $\mathcal{T}(\Gamma)$ .

142 It is worth noting [6] that, in general, the size  $|\mathcal{T}(\Gamma)|$  of the testing set  $\mathcal{T}(\Gamma)$  can be exponential  
143 in the representation of  $\tau$ . However, it has been proven [5, Theorem 3.1] that for *bounded-*  
144 *utilization* task systems —i.e., systems  $\Gamma$  that fulfill the additional requirement that  $\sum_{\tau_i \in \Gamma} U_i \leq c$   
145 for some fixed constant  $c$  strictly less than 1— it is sufficient to check a smaller testing set with  
146 pseudo-polynomial cardinality relative to the representation of  $\Gamma$ , consisting of all values of the  
147 form  $L \equiv (k \times T_i + D_i)$  not exceeding

$$148 \quad \min \left( P, \max \left( D_{\max}, \frac{1}{1 - U} \cdot \sum_{i=1}^n U_i \cdot (T_i - D_i) \right) \right) \quad (3)$$

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

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

156 Unfortunately, the general problem is NP-hard in the strong sense [18, 21, 20], and the  
 157 bounded-utilization variant is NP-hard in the ordinary sense [19]. Therefore, it is unlikely that we  
 158 will discover more efficient schedulability tests.

## 159 2.2 Limited-Preemption Scheduling

160 The limited-preemption sporadic task model, as introduced by Baruah et al. [7], adds to the task  
 161 specification  $\tau_i = (C_i, D_i, T_i, \beta_i)$  a *chunk-size* parameter  $\beta_i$  in addition to the regular parameters  
 162  $C_i$ ,  $D_i$ , and  $T_i$ . This parameter  $\beta_i$  indicates that each job of task  $\tau_i$  may need to execute  
 163 non-preemptively for up to  $\beta_i$  time units.

164 To schedule tasks in the limited-preemption sporadic task model, the limited-preemption  
 165 EDF scheduling algorithm was proposed [7, 12]. Like its preemptive counterpart, the limited-  
 166 preemption EDF algorithm prioritizes jobs based on their (absolute) deadlines. If a job of task  $\tau_i$   
 167 with remaining execution time  $e$  is executing and a new job with an earlier deadline arrives, then  
 168  $\tau_i$ 's job may execute for an additional  $\min(e, \beta_i)$  time units before incurring a preemption.

169 Baruah and Bertogna [7, 12] showed that a task system is *not* schedulable under the limited-  
 170 preemption EDF model if and only if either of the following conditions are true:

$$171 \quad \exists L : L \geq 0 : \sum_{\tau_i \in \Gamma} \text{DBF}_i(L) > L \quad (4)$$

172 OR

$$173 \quad \exists \tau_i : \exists L : 0 \leq L < D_i : \beta_i + \sum_{\tau_j \in \Gamma, j \neq i} \text{DBF}_j(L) > L \quad (5)$$

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

$$177 \quad \forall L, \quad \left[ \left( \sum_{\tau_i \in \Gamma} \text{DBF}_i(L) \right) + \overbrace{\max_{\{\tau_i | D_i > L\}} \{\beta_i\}}^{\text{(blocking due to limited preemption)}} \leq L \right] \quad (6)$$

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

184 In this extension to the prior work in [8], we use a continuous-time representation of the blocking  
 185 term in Equation 6. The prior work used a discrete-time representation,  $\max_{\tau_i | D_i > L} \beta_i - 1$ , which  
 186 requires both task periods *and execution times* to be represented as integers. This introduces  
 187 several challenges. First, it is more difficult to reason about the effect of inserting preemption

188 points; blocking times (chunk sizes) must also be represented as integers, but the number of  
 189 “chunks” might not evenly divide the execution time. Second, there is a tradeoff when choosing  
 190 the precision at which to represent execution time units. If the unit of time is coarse, then the  
 191 execution time of each phase and its corresponding startup/teardown time must be rounded up.  
 192 If the unit of time is very short —such as a single processor tick— to achieve higher precision,  
 193 then the testing set grows rapidly as the representations of the task periods become larger. By  
 194 using continuous time, this extension removes these limitations, and modifies the expression to  
 195 allow execution times to take any non-negative real value.

196 Based on this observation, the Processor Demand Analysis (PDA) algorithm has been extended  
 197 to apply to limited-preemption systems as well [7]. The extension is straightforward: Equation 6  
 198 replaces Equation 2 in the algorithm, and the algorithm proceeds as usual.

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

$$200 \quad 0 \leq L < \min \left( D_{\min}, \max_{\{\tau_i\}} \{\beta_i\} \right) \quad (7)$$

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

### 204 **3 The Corrected Limited-Preemption EDF Schedulability Condition**

205 In this section, we present a correction to the condition of Baruah and Bertogna [7, 12] for EDF  
 206 schedulability of limited-preemption tasks.

#### 207 **3.1 The Problem**

208 As discussed in the prior section, in [7, 12], Baruah and Bertogna claimed as a necessary and  
 209 sufficient condition for scheduling a limited-preemption sporadic task system upon a single  
 210 preemptive processor using EDF that

$$211 \quad \forall L, \left[ \left( \sum_{\tau_i \in \Gamma} \text{DBF}_i(L) \right) + \overbrace{\max_{\{\tau_i | D_i > L\}} \{\beta_i\}}^{\text{(blocking due to limited preemption)}} \leq L \right] \quad (8)$$

212 where  $\beta_i$  is the blocking due to limited preemption induced by task  $\tau_i$ .

213 From the definition of the demand bound function  $\text{DBF}_i(L)$  in Equation 1, we observe that  
 214  $\text{DBF}_i(L) = 0$  for  $L < D_i$ . Then for  $L < D_{\min}$  (i.e.,  $L < D_i$  for all tasks  $\tau_i$ ), the above condition  
 215 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  
 216 simplified to  $L \geq \max_{\tau_i} \{\beta_i\}$ .

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

#### 222 **3.2 The Correction**

223 As this is obviously not true – i.e., there **are** limited-preemption task sets that are schedulable by  
 224 EDF, we now set out to correct the above condition. We do so by pointing out a subtlety to the  
 225 **proof** in [7] 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. Additionally, for each job  $\tau_i$ , it defines  $q_i$  as representing an upper bound on the time for which  $\tau_i$  can execute non-preemptively. In this system, there is exactly one job with a deadline after  $t_f$ ; we let  $\tau_j$  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 [7, Equation 4] states that

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

As  $q_j$  is a bound on the non-preemptive execution time, the proof in [7] then claims that  $t_2 - t_1 \leq q_j$ . 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 [7]. 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  $\tau_j$  will therefore be zero at  $L$  and we can rewrite the condition as:

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

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

$$230 \left( \sum_{\tau_i \in \Gamma} \text{DBF}_i(L) \right) + \min \left( L, \max_{\{\tau_i | D_i > L\}} \{\beta_i\} \right) \leq L \quad (9)$$

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

## 235 4 Systems of Multi-Phase Secure (MPS) Sporadic Tasks

236 In this section we extend the sporadic task model to consider the setting where the workload  
 237 of each task comprises an ordered sequence of different *phases*, with each phase required to use  
 238 a different security mechanism.<sup>2</sup> Therefore, switching between phases or between jobs incurs  
 239 some *teardown/startup overhead*, which translates to an additional execution duration. We  
 240 are given a system of multiple such independent tasks that are to be scheduled upon a shared  
 241 preemptive processor. If an executing task is preempted within a phase, the assumption that the  
 242 tasks in the system are independent of one another implies that we must conservatively assume

<sup>2</sup> Note that we consider a unique *combination* of security mechanisms to be its own security mechanism. Thus, a portion of a task subject to multiple overlapping security mechanisms would execute in its own phase.

243 that the preempting task may be executing using a different security mechanism; hence, the  
 244 teardown/startup overhead may be incurred again. In some systems, the cost of a preemption  
 245 may be less than the full startup/teardown cost of a phase; this can be accounted for by adding  
 246 any additional cost to the execution time of the phase. When a phase of a task is selected for  
 247 execution we assign it responsibility for taking care of both the startup that must happen at that  
 248 point in time, and the subsequent teardown that occurs when it either completes execution or is  
 249 preempted by a higher-priority task. Hence a phase with execution duration  $c$  that is preempted  $k$   
 250 times is responsible for (and hence should have been budgeted for) a total execution duration of

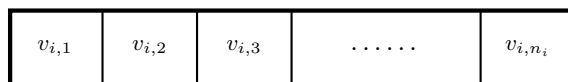
$$251 \quad c + (k + 1) \times (\text{startup cost} + \text{teardown cost}) \quad (10)$$

252 **A Motivating Example.** To motivate the MPS task model, consider the case of *trusted execution*  
 253 *environments* (TEEs), a hardware feature that provides strong isolation of code and data. Although  
 254 this isolation has significant security benefits, broadly-used TEE implementations such as OP-  
 255 TEE [1] are sometimes limited to using only a small fraction of the available system memory [2]  
 256 and code running in a TEE has limited access to common system APIs. In addition, placing more  
 257 functionality inside a TEE increases the probability that an unidentified vulnerability compromises  
 258 the isolated TEE environment; a number of works based on TEEs have cited minimization of the  
 259 trusted computing base (TCB) as a key design concern [28, 32, 40]. A fine-grained minimization  
 260 of the code placed in a TEE can lead to the existence of tasks that span multiple “worlds” – for  
 261 example, a task may begin execution in the normal world (i.e. no TEE), switch to the secure  
 262 world (executing in the TEE) to perform a sensitive cryptographic operation, and then return to  
 263 the normal world to complete its work. In some cases, switching across worlds can also be used as  
 264 a workaround for TEE memory limitations. For example, large neural networks may be executed  
 265 securely by copying a single layer into the TEE, computing and storing an intermediate result,  
 266 and then returning to the normal world to retrieve the next layer [4].

267 However, switching between worlds induces TEE startup and teardown costs, the impact  
 268 of which varies depending on the choice of processor and TEE software implementation; one  
 269 implementation based on an Arm Cortex-M development board saw costs on the order of micro-  
 270 seconds [33], while an implementation using OP-TEE on a Raspberry Pi with a Cortex-A processor  
 271 saw overheads ranging from hundreds of microseconds to tens of milliseconds [35]. A scheduling  
 272 approach that does not consider these significant context-switching costs may preempt execution  
 273 for higher-priority jobs indiscriminately, resulting in missed deadlines due to the overheads incurred  
 274 by frequent startup/teardown.

## 275 4.1 Task Model

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



281  
 282 As previously discussed, we assume that successive phases are required to execute using different  
 283 security mechanisms (i.e.,  $v_{i,j}$  and  $v_{i,j+1}$  execute using different security mechanisms for all  $j$ ,  
 284  $1 \leq j < n_i$ ). Let  $c(v_{i,j})$  denote the WCET of phase  $v_{i,j}$ , and  $q(v_{i,j})$  denote the sum of the startup



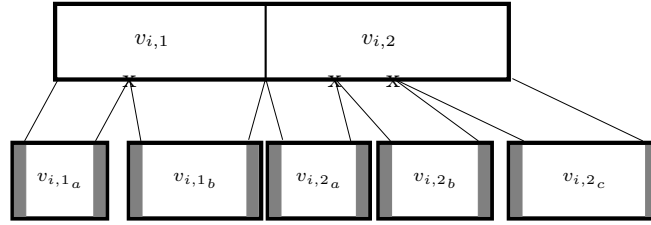
285 cost and the teardown cost associated with the security mechanism within which phase  $v_{i,j}$  is to  
 286 execute. The aggregate WCET of all phases of this task during its execution is thus given by the  
 287 following expression

$$288 \sum_{j=1}^{n_i} (c(v_{i,j}) + q(v_{i,j}))$$

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

$$292 \sum_{j=1}^{n_i} (c(v_{i,j}) + (k_j + 1) \times q(v_{i,j}))$$

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



## 297 4.2 Overview of Approach

298 Given a task system  $\Gamma$  comprising multiple independent MPS tasks to be scheduled upon a  
 299 single preemptive processor, we will first execute a schedulability analysis algorithm to determine  
 300 whether this system is schedulable, i.e., whether we can guarantee to schedule it to always meet all  
 301 deadlines, despite the costs incurred by startup/teardown. This schedulability analysis algorithm  
 302 essentially constructs a limited-preemption task  $\hat{\tau}_i$  corresponding to each task  $\tau_i$ , and determines  
 303 whether the resulting limited-preemption task system can be scheduled by the limited-preemption  
 304 EDF scheduling algorithm [7, 12] to always meet all deadlines. If so, then during run-time the  
 305 original task system is scheduled using the limited-preemption EDF scheduling algorithm, with  
 306 chunk-sizes as determined for the corresponding constructed limited-preemption tasks. We point  
 307 out that if a chunk-size  $\beta_i$  is determined for the limited-preemption task  $\hat{\tau}_i$ , then the  $j$ 'th phase  
 308 of  $\tau_i$ '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  
 309 would experience at most  $(\lceil c(v_{i,j})/(\beta_i - q(v_{i,j})) \rceil - 1)$  preemptions); equivalently, the cumulative  
 310 WCET of all the phases of each of task  $\tau_i$ 's jobs will be no more than

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

311 **Assumption of Fixed-Preemption Points.** We make the conservative assumption that once  
 312 the chunk-size is determined for each task then the preemption points in the code are statically  
 313 determined prior to run-time. That is, once the chunk-size  $\beta_i$  is determined for a task  $\tau_i$ , a  
 314 preemption is statically inserted into the task's code after the code has executed non-preemptively  
 315 for no more than  $\beta_i$  time units; this is referred to as the *fixed-preemption point model* [43]. Once  
 316  $\tau_i$ 's program reaches this statically-placed preemption point, the security mechanism for the task's  
 317 current phase must

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

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

### 331 4.3 The Schedulability Test

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

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

$$340 \quad \widehat{C}_i \stackrel{\text{def}}{=} \sum_{j=1}^{n_i} \left( c(v_{i,j}) + \text{cnt}(v_{i,j}) \times q(v_{i,j}) \right) \quad (11)$$

341 where the second term within the summation represents the maximum preemption overhead (the  
342 startup cost plus the teardown cost) that is incurred by the  $j$ ’th phase of task  $\tau_i$ .

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

$$350 \quad \beta_i \leftarrow \max_{j=1}^{n_i} \{c(v_{i,j}) + q(v_{i,j})\} \quad (12)$$

351 In this manner, we have instantiated the parameters for one limited-preemption task  $\widehat{\tau}_i$  corres-  
352 ponding to each task  $\tau \in \Gamma$ . We must now check whether the limited-preemption task system  
353  $\widehat{\Gamma} = \{\widehat{\tau}_1, \widehat{\tau}_2, \dots, \widehat{\tau}_N\}$  so obtained is schedulable using the limited-preemption EDF scheduling  
354 algorithm [7, 12]. We do so by checking whether Equation 9 holds for values of  $t$  in the testing  
355 set  $\mathcal{T}(\widehat{\Gamma})$ , considered in *increasing order*; we motivate this ordering below in (5). First, we split  
356  $\mathcal{T}(\widehat{\Gamma})$  into two sets,  $\mathcal{T}(\widehat{\Gamma})_1$  containing all values up to and including  $D_{\max} \equiv \max_{\tau_i} D_i$ , and  $\mathcal{T}(\widehat{\Gamma})_2$   
357 containing all values thereafter. Then, we initialize  $t_d$  to denote the smallest value in  $\mathcal{T}(\widehat{\Gamma})_1$ , and  
358 perform the following steps.

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

---

**Input:**  $(\Gamma)$

```

1 for each task  $\tau_i \in \Gamma$  do //Initially, assume that no preemption is needed
2   for  $j \leftarrow 1$  to  $n_i$  do
3     cnt( $v_{i,j}$ )  $\leftarrow 1$ 
4      $\beta_i \leftarrow \max_{1 \leq j \leq n_i} (c(v_{i,j}) + q(v_{i,j}))$ 
5 //The testing set  $\mathcal{T}(\Gamma)_1$  is all  $t \equiv D_i + k \cdot T_i$ ,  $t \leq D_{\max}$  for some task  $\tau_i$  and some  $k \in \mathbb{N}$ .
6 for  $t_d$  iterating in increasing order over  $\mathcal{T}(\Gamma)_1$  do
7   Compute  $\Delta(t_d)$  as per Eqn 13 //This represents the slack in the schedule at  $t_d$ 
8   if  $\Delta(t_d) < 0$  then //Check whether previously-assigned chunk sizes causes deadline miss
9     return THE SYSTEM IS NOT SCHEDULABLE
10  for each  $\tau_i$  for which  $D_i > t_d$  do
11    if  $(\beta_i > \Delta(t_d))$  then //Must reduce the value of  $\beta_i$ 
12       $\beta_i \leftarrow \Delta(t_d)$ 
13      for  $j \leftarrow 1$  to  $n_i$  do //For each phase of  $\tau_i$ , ensure that it doesn't block too much
14        if  $\beta_i > q(v_{i,j})$  then //There is sufficient time in chunk to do task execution
15          cnt( $v_{i,j}$ )  $\leftarrow \min_{\kappa \in \mathbb{N}}$  such that  $\frac{c(v_{i,j})}{\kappa} + q(v_{i,j}) \leq \beta_i$  //Break  $c(v_{i,j})$  into small enough
16          pieces
17        else
18          return THE SYSTEM IS NOT SCHEDULABLE
19 if system utilization  $> 1$  then
20   return THE SYSTEM IS NOT SCHEDULABLE
21 if all tasks have implicit deadlines  $D_i = T_i$  then
22   return THE SYSTEM IS SCHEDULABLE
23 //For systems of constrained-deadline tasks, the testing set  $\mathcal{T}(\Gamma)_2$  is all  $t \equiv D_i + k \cdot T_i$ ,
24    $D_{\max} < t$  and not exceeding the bound defined in Expression 3 for some task  $\tau_i$  and some  $k \in \mathbb{N}$ .
25 for  $t_d$  iterating in increasing order over the testing set  $\mathcal{T}(\Gamma)_2$  do
26   Compute  $\Delta(t_d)$  as per Eqn 13
27   if  $\Delta(t_d) < 0$  then
28     return THE SYSTEM IS NOT SCHEDULABLE
29 return THE SYSTEM IS SCHEDULABLE

```

---

- 359 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  
360 this step.
- 361 2. If Equation 9 is violated for  $t_d$  and the first term in the LHS of Equation 9 is  $> t_d$ , then we  
362 conclude that THE SYSTEM IS NOT SCHEDULABLE and return.

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

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

$$371 \quad \Delta(t_d) \stackrel{\text{def}}{=} t_d - \sum_{\widehat{\tau}_k \in \widehat{\Gamma}} \text{DBF}(\widehat{\tau}_k, t_d) \quad (13)$$

## 42:12 Limited-Preemption EDF Scheduling for Multi-Phase Secure Tasks

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

$$376 \quad \text{cnt}(v_{i,j}) \leftarrow \min_{\kappa \in \mathbb{N}} \text{such that } \frac{c(v_{i,j})}{\kappa} + q(v_{i,j}) \leq \Delta(t_d) \quad (14)$$

377 That is, we reduce blocking in order to satisfy Equation 9 for  $t_d$  by potentially increasing the  
 378 number of preemptions (and thereby incurring additional teardown/startup overhead). In  
 379 prior work [8], we assumed that tasks had integer execution times and hence used  $\beta_i - 1$  as  
 380 the blocking term; here, we use continuous time and therefore do not subtract a time segment.

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

$$384 \quad \widehat{C}_i \leftarrow \sum_{j=1}^{n_i} \left( c(v_{i,j}) + \text{cnt}(v_{i,j}) \times q(v_{i,j}) \right)$$

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

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

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

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

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

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

408 **Computational Complexity.** The number of iterations of the *for-loops* of Lines 5 and 21  
 409 dominate the computational complexity; the other loops in the algorithm are polynomial in the

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

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

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

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

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

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

438 **Initialization:** Initially  $t_d$  equals  $D_{\min}$ ; the minimum deadline is the first non-zero value of the  
 439 DBF function for all tasks. Thus, Statement 1 is true since at all prior timepoints the first term of  
 440 Equation 9 is zero and Equation 9 reduces to  $L \leq L$  per the arguments in Section 3.2. Statement 2  
 441 is also vacuously true since  $\beta_1$  is set to its largest possible value  $\max_{1 \leq j \leq n_1} (c(v_{1,j}) + q(v_{1,j}))$  by  
 442 the previous loop and does not affect schedulability as  $\tau_1$  cannot block any other task (by nature  
 443 of having the smallest relative deadline).

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

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

455 Statement 2 follows from the last observation of the previous paragraph and by the assumption  
 456 that  $\beta$  values for all  $\tau_i$  with  $D_i \leq t_c$  are set to their maximum value by assumption and cannot

## 42:14 Limited-Preemption EDF Scheduling for Multi-Phase Secure Tasks

457 change at  $t_c$  or after. Therefore, the  $\widehat{C}_i$  values that contribute to the DBF in Equation 9 at  $t_c$  are  
 458 as small as possible. It therefore follows that any  $\tau_j$  with  $t_c \leq D_j \leq t_d$  has either already had its  
 459  $\beta_j$  value set to the largest possible to satisfy Equation 9 for some  $t < t_c$  or has its value set to the  
 460 largest possible to satisfy Equation 9 at  $t_c$ .

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

468 **► Lemma 2.** *In an implicit-deadline task system, Equation 9 will be satisfied at all points*  
 469  *$L > D_{\max}$  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  $\tau_i$ ,

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

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:

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

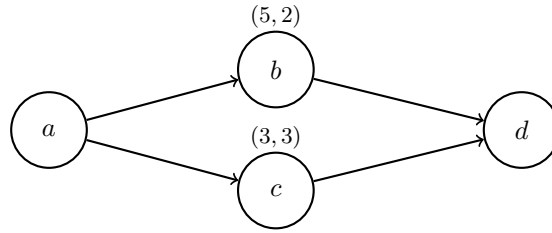
470 which implies that  $U > 1$ .  $\square$

471 **► Theorem 1.** *If the schedulability test returns THE SYSTEM IS SCHEDULABLE, then assignment*  
 472 *of  $\beta_i$  values by the algorithm in Algorithm 1 ensures that the task system meets all deadlines when*  
 473 *scheduled by limited-preemption EDF.*

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

480 Since Equation 9 is sufficient (and necessary for the fixed-preemption model), the task system  
 481  $\Gamma$  is schedulable by limited-preemption EDF when the assigned chunk-sizes  $\beta_i$  are used.  $\square$

482 **► Theorem 2.** *If the schedulability test returns THE SYSTEM IS NOT SCHEDULABLE, then there*  
 483 *does not exist an assignment of  $\beta_i$  values such that Equation 6 is satisfied.*



■ **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.

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

491 If THE SYSTEM IS NOT SCHEDULABLE is returned while processing a testing-set interval  $t_d$  in  
 492  $\mathcal{T}(\hat{\Gamma})_2$ , then Equation 9 is false for some  $t_d \in \mathcal{T}(\hat{\Gamma})_2$ . The right-side term of equation 9 considers  
 493 only tasks where  $D_i > L$ ; as  $\mathcal{T}(\hat{\Gamma})_2$  begins past  $D_{\max}$ , there can be no tasks matching this  
 494 condition and therefore no assignment of  $\beta_i$  that would reduce the right-side term. Per Statement  
 495 2 of Lemma 1, each  $\beta_i$  has already been maximized when considering  $\mathcal{T}(\hat{\Gamma})_1$ ; reducing some  $\beta_i$   
 496 can only increase the number of preemption points required and therefore increase the DBF of  
 497 some task in the left-side term of Equation 9. Therefore, there is no alternative assignment of  $\beta_i$ 's  
 498 that would reduce the left side of Equation 9 and cause the system to become schedulable.  $\square$

## 499 5 Systems of Conditional MPS Sporadic Tasks

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

516 **5.1 Model**

517 We now provide a more formal description of the *conditional MPS sporadic task model*. Each task  
 518  $\tau_i$  is characterized by a 3-tuple  $(G_i, D_i, T_i)$  where  $D_i$  and  $T_i$  are the relative deadline and period,  
 519 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  
 520 computation, which must execute using a specified security mechanism. The interpretation of each  
 521 edge  $(v_{i,j}, v_{i,k}) \in E$  depends upon the outdegree (i.e., the number of outgoing edges) of vertex  $v_{i,j}$ :

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

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

532 In the original model presented in [8], a simplifying assumption was made that teardown and  
 533 setup costs are always incurred when transitioning between any two jobs, regardless of whether  
 534 they use the same or different security mechanisms. This assumption was conservative, as it  
 535 did not account for cases where both jobs use the same security mechanism, in which case the  
 536 teardown and setup costs would not actually be incurred.

537 **5.2 A Subtlety**

538 A subtle issue arises when dealing with conditional code, which was not present in the consideration  
 539 of linear workflows in Section 4. Specifically, during the execution of conditional code, it is not  
 540 known at compile time which branch will be taken at runtime, and different invocations may  
 541 take different paths. In hard real-time systems, it is necessary to ensure that tasks meet their  
 542 deadlines under all possible runtime conditions. Thus, during pre-runtime schedulability analysis,  
 543 a conservative approach is adopted, assuming that each invocation of the task follows the ‘longest’  
 544 path—the one with the maximum cumulative execution requirement.

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

549 ■ If the chunk size  $\beta_i$  for this task were  $\geq 7$ , then both branches can be executed non-preemptively.  
 550 The upper branch incurs a cost of  $5 + 2 = 7$  while for the lower branch the cost is  $3 + 3 = 6$ ;  
 551 therefore, the upper branch is the computationally more expensive one.

552 ■ Now suppose  $\beta_i = 4$ . Then the upper branch may need to execute in  $\lceil 5/(4-2) \rceil$  or 3 contiguous  
 553 pieces, for a cumulative cost of  $5 + 3 \times 2 = 11$ . The lower branch may need to execute in  
 554  $\lceil 3/(4-3) \rceil$  or 3 contiguous pieces, for a cumulative cost of  $3 + 3 \times 3 = 12$ , and is hence the  
 555 more expensive branch.

556 This example illustrates that the computationally most expensive path through a DAG that  
 557 represents conditional execution depends upon the value of the chunk size parameter (the  $\beta_i$   
 558 parameter of the task). As we saw in Section 4, the approach to scheduling MPS sporadic tasks  
 559 has been to convert each task to a limited-preemption sporadic task. The procedure for doing so  
 560 (Algorithm 1) repeatedly changes the values of the  $\beta_i$  parameters of the tasks. As we adapt the



561 techniques of Section 4 to schedulability analysis of conditional MPS tasks, it is important to note  
 562 that the necessary improvements to handle conditional execution, including the correct analysis  
 563 of teardown and setup costs, have already been addressed in a separate extension paper [39]. In  
 564 that work, the unnecessary conservatism in the assumption that teardown and setup costs are  
 565 always incurred during transitions between jobs, regardless of whether they use the same security  
 566 mechanism is eliminated. This refinement leads to a more precise schedulability analysis, ensuring  
 567 that only the relevant overheads are considered and optimizing the overall analysis of conditional  
 568 MPS tasks. Thus, while this paper focuses on the linear task model and its optimization, the  
 569 more accurate treatment of conditional execution is fully covered in [39].

## 570 **6 Empirical Evaluation**

571 In the previous sections, we have developed algorithms to introduce preemptions into the execution  
 572 of the phases of multi-phase secure tasks. While these preemptions reduce the blocking that  
 573 higher-priority phases/jobs experience at runtime, they come at a cost – increased overhead due  
 574 to the additional teardown/startup costs that must be performed before and after every inserted  
 575 preemption. Thus, while the limited-preemption approach proposed in this paper theoretically  
 576 dominates the non-preemptive approach (i.e., each phase is executed non-preemptively and  
 577 preemptions are permitted only between phases of a task), it is unclear how much schedulability  
 578 improvement *on average* we can expect a system designer to obtain from using our proposed  
 579 approach.

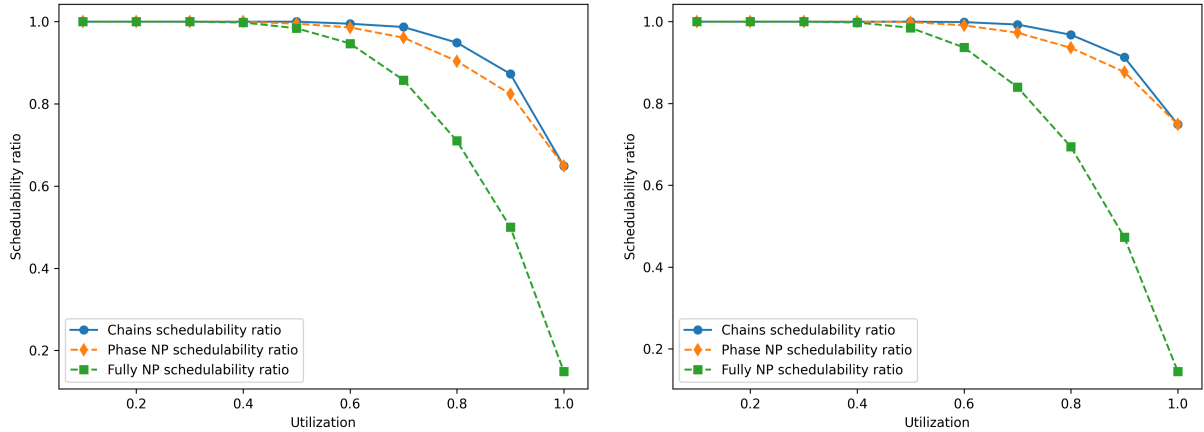
580 In this section, we provide an empirical analysis on that topic via the application of the  
 581 schedulability tests of Section 4 over synthetically-generated MPS task systems. We compare the  
 582 proposed schedulability tests with existing limited-preemption scheduling where each phase of a  
 583 task is executed fully non-preemptively.

584 By using the refined algorithms presented in this work that permit pseudo-polynomial running  
 585 times for bounded utilization task sets, and correcting implementation issues identified in our  
 586 earlier work [8], we were able to extend our experiments to a broader parameter space and larger  
 587 task sets. We also directly evaluate the running times of our algorithms and implementation, and  
 588 compare them to the original versions from [8].

589 As before, we limit our scope to evaluating linear task sequences; we refer readers interested in  
 590 further evaluation, analysis, and refinement of the conditional model to [39].

### 591 **6.1 Experimental Setup**

592 The evaluation was conducted using a C++ simulation. All tests were performed on a  
 593 server with two Intel Xeon Gold 6130 (Skylake) processors running at 2.1 GHz, and with 64GB of  
 594 memory. Multiple task sets were evaluated in parallel; each task set was given a single thread  
 595 on which to run. We evaluate task sets for many parameter variations, including the task count,  
 596 number of phases per task, utilization, and task periods. In some cases, the number of phases per  
 597 task is fixed; where it is not fixed, we select the number of phases for each task following a uniform  
 598 distribution in the given range. Task periods are selected either from a uniform distribution  
 599 within the period range specified for each test or, where specified below, a log-uniform distribution.  
 600 For each combination of fixed parameters, we generate 1000 random task systems. For each  
 601 system, we use the UUniFast algorithm [15], first to assign a total utilization  $\frac{C_i}{T_i}$  to each task  
 602 and then to distribute the task’s total allotted time  $C_i$  between the execution times  $c_{(v_{i,j})}$  and  
 603 startup/teardown overhead  $q_{(v_{i,j})}$  for each of its phases. We note that compared to the prior work  
 604 in [8] that this paper extends, we have adjusted this distribution to be more consistent and to be  
 605 independent of the number of phases in the task. We then evaluate each task set against three



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

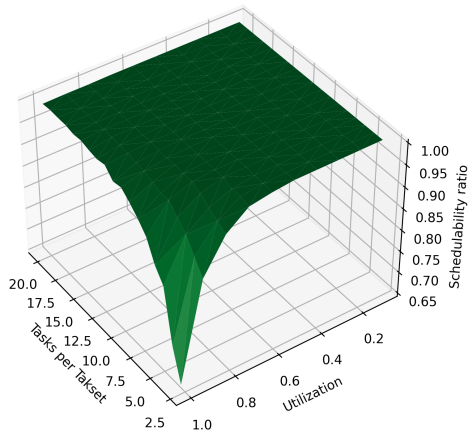
606 algorithms: *chains*, the algorithm presented in section 4, *phase NP*, in which there is a static  
 607 preemption point between each phase but no additional preemption points can be inserted, and  
 608 *fully NP*, in which the entire task runs as a single non-preemptive chunk.

609 We evaluate both implicit-deadline and constrained-deadline task systems. In an implicit-  
 610 deadline system,  $D_i = T_i$  for each task. In a constrained-deadline system, we choose a random  
 611 value for each  $D_i$  that is uniformly distributed between the task’s execution time and its period  
 612  $T_i$ . For implicit-deadline systems, we evaluate  $U$  at increments of 0.1 in  $[0, 1]$ . We note that,  
 613 compared to the prior version of this work in [8], the improvement to the testing set described  
 614 in Section 4 allows us to evaluate these systems in a reasonable amount of time even for large  
 615 numbers of tasks; to understand the impact of this optimization, we also test the exponential set  
 616 presented in the original version on systems with 8 tasks or fewer and compare their execution  
 617 times. For constrained-deadline systems, the testing set is bounded by a term that is determined  
 618 in part by a factor  $\frac{1}{1-U}$ ; as  $U \rightarrow 1$ , the testing set size becomes very large and becomes infeasible  
 619 to evaluate. We therefore limit our evaluation of constrained-deadline systems to utilizations in  
 620  $[0, 0.9]$ .

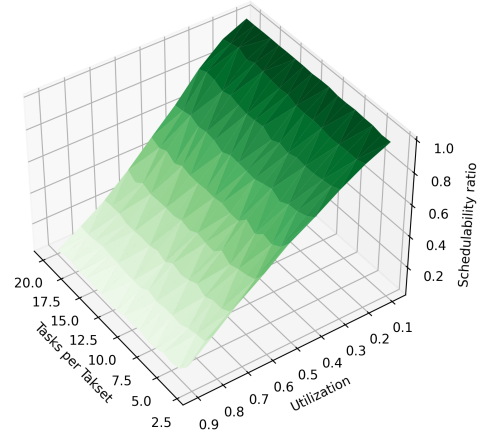
## 621 6.2 Schedulability Analysis Results

622 **Schedulability of Implicit-Deadline Systems.** Figure 2 shows the schedulability ratio of each  
 623 of the three tests described above — chains, phase NP, and fully NP. For each test, tasks with  
 624 low utilization are all schedulable, but schedulability decreases as the utilization of the system  
 625 increases. By inserting additional preemption points, the chains algorithm is able to obtain a  
 626 schedulability improvement over a phase NP approach on these higher-utilization tasks. Once  
 627 utilization reaches 1, it is not possible to insert any preemption points, as inserting a preemption  
 628 point would increase the startup/teardown overhead and cause the utilization to exceed 1; therefore,  
 629 the chains algorithm does not lead to a schedulability improvement. Tasks with 1-6 phases have a  
 630 slightly higher schedulability ratio than tasks with 1-4 phases; we explore this effect more in the  
 631 paragraph below.

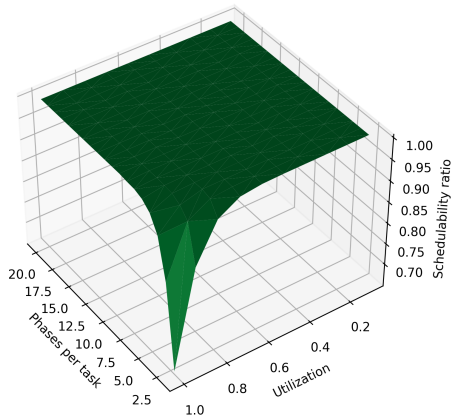
632 Figure 3 shows the impact of various parameters of the task system on the schedulability ratio  
 633 using the chains algorithm and periods of 10–30 time units. In Figure 3a, tasks are randomly  
 634 assigned 1-4 phases, while the utilization and tasks per taskset are varied. In Figure 3c, the



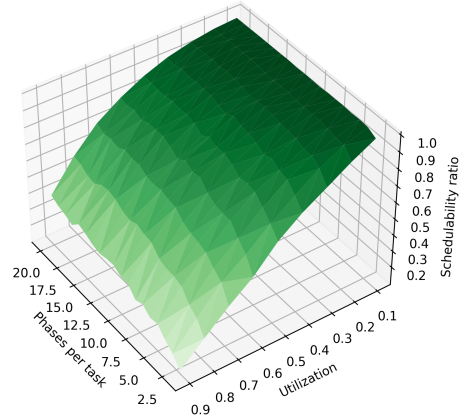
(a) Implicit deadlines, 1–4 phases.



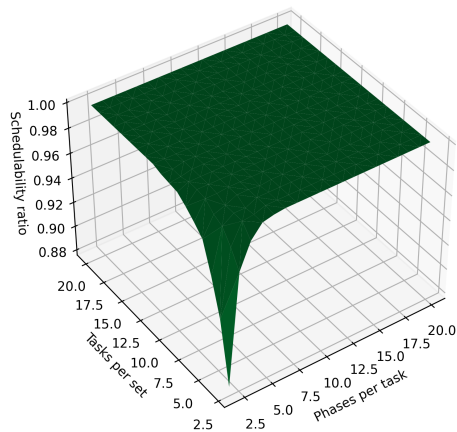
(b) Constrained deadlines, 1–4 phases.



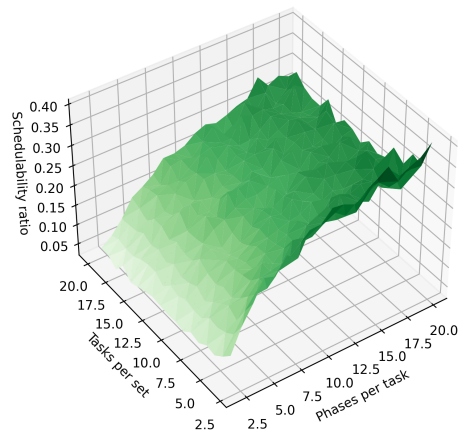
(c) Implicit deadlines, 3 tasks per set.



(d) Constrained deadlines, 3 tasks per set.

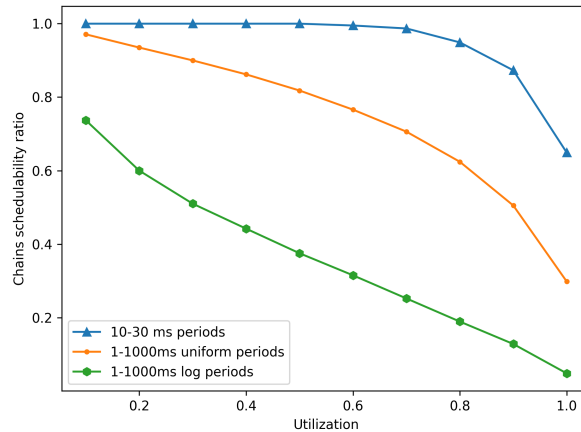


(e) Implicit deadlines, utilization 0.9.



(f) Constrained deadlines, utilization 0.9.

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



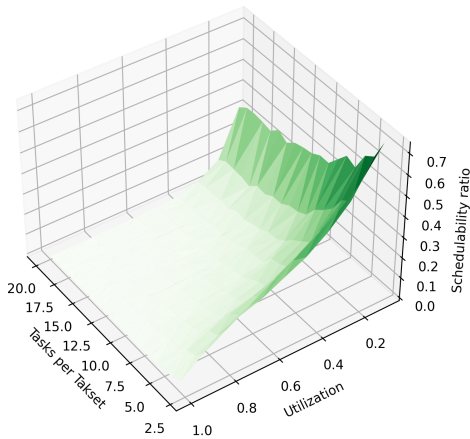
■ **Figure 4** Comparison of the effect on schedulability of changing the period range of the generated tasks. Each set contains 3 implicit-deadline tasks with 1–4 phases.

635 number of tasks is held constant at 3, and each taskset is assigned a fixed number of phases,  
 636 varying from 2 to 20 phases per task. In Figure 3e, the utilization is held constant at 0.9. As  
 637 in Figure 2, increasing utilization reduces the schedulability ratio. Here, it is clearly visible that  
 638 increasing the number of tasks or number of phases per task improves the schedulability ratio.  
 639 When these parameters are increased, the system’s total execution time is divided among more  
 640 tasks or among more phases, so that each task has a lower blocking time. The reduction in  
 641 blocking time makes the system more likely to be schedulable.

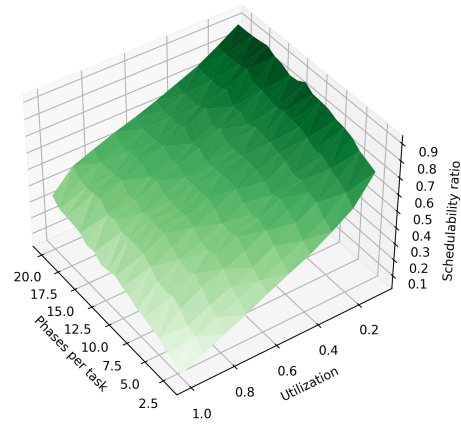
642 For completeness, we also evaluate the performance of the algorithm on a wider 1–1000 time  
 643 unit period range, using both a uniform distribution of periods within this range, as well as the  
 644 log-uniform distribution recommended in [22]. Figure 4 shows how the schedulability of these task  
 645 sets compares to the tasksets with 10–30 unit periods. The schedulability of the sets with the  
 646 wider period range is much lower; in particular, tasksets containing a task with a small period are  
 647 less likely to be schedulable. We hypothesize that this is due to these high-frequency tasks having  
 648 less ability to expand their execution time as preemption points are inserted. This trend is also  
 649 apparent in Figure 5; unlike tasks with 10–30 unit periods, increasing the number of tasks increases  
 650 the probability of generating a high-frequency task and therefore causes the schedulability ratio to  
 651 decrease.

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

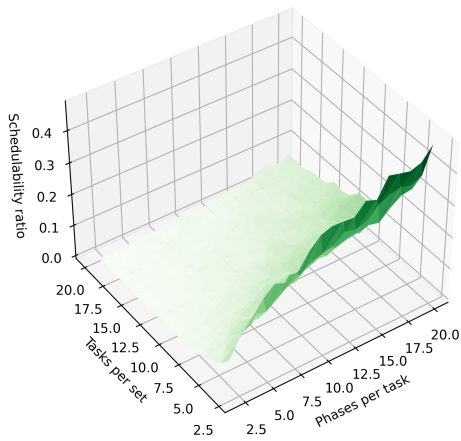
659 The right column of Figure 3 shows the effect of varying different taskset parameters for the  
 660 constrained-deadline task sets, which otherwise have the same parameter configurations as the  
 661 implicit-deadline systems. For constrained-deadline tasks, the schedulability ratio is generally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



(a) Implicit deadlines, 1–4 phases.



(b) Implicit deadlines, 3 tasks per set.



(c) Implicit deadlines, utilization 0.9.

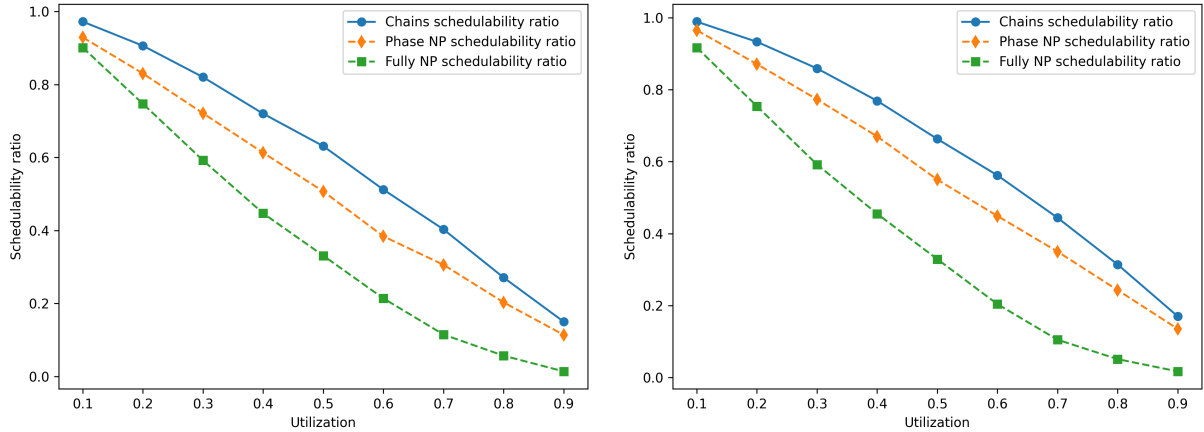
■ **Figure 5** Schedulability ratio of chains algorithm when varying taskset parameters. All tasks have periods selected from 1–1000 time units using a log-uniform distribution.

666 system will be schedulable.

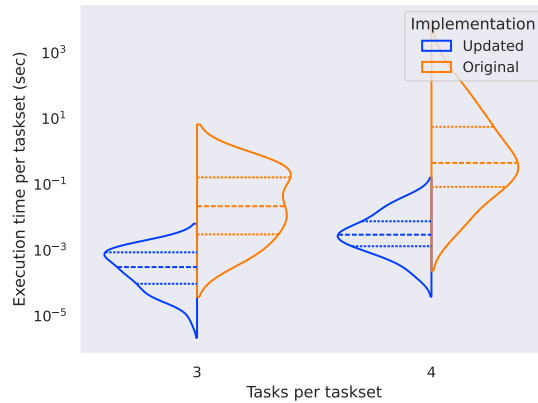
### 667 6.3 Runtime Performance

668 The implementation of the algorithm in our prior work [8] could only evaluate sets of a few tasks  
 669 in a reasonable amount of time. In this work, we rewrite the original Python-based simulation  
 670 using C++, and correct an implementation issue with the original construction of the hyperperiod-  
 671 bounded testing set. These changes lead to significant performance improvements on their own.  
 672 In Figure 7 we randomly generate and test 50 task sets with either 3 or 4 tasks each; the original  
 673 implementation is compared to the rewritten one, which is configured to test points in the full,  
 674 exponentially-sized testing set. For sets of 3 tasks, the median execution time improves by  
 675 approximately 70× and the mean execution time improves by around 250×. For sets of 4 tasks,  
 676 the median improves by around 150× and the mean execution time by more than 3500×. These

## 42:22 Limited-Preemption EDF Scheduling for Multi-Phase Secure Tasks



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

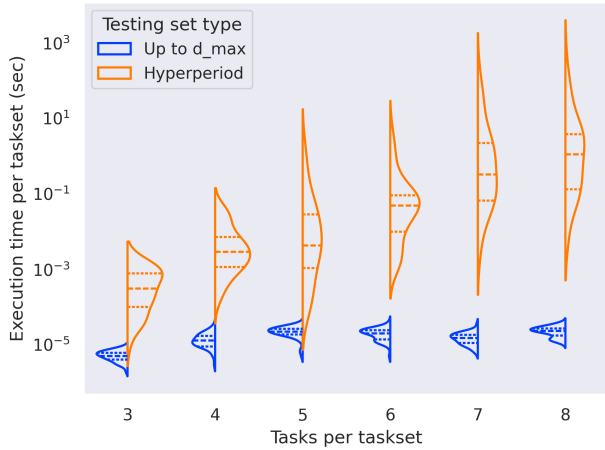


■ **Figure 7** Comparison of the time needed to determine schedulability for task sets with 1-4 phases, utilization of 0.9, and periods of 10-30. The left side is the rewritten C++ implementation; the right side is the original implementation from [8]. Note the log scale.

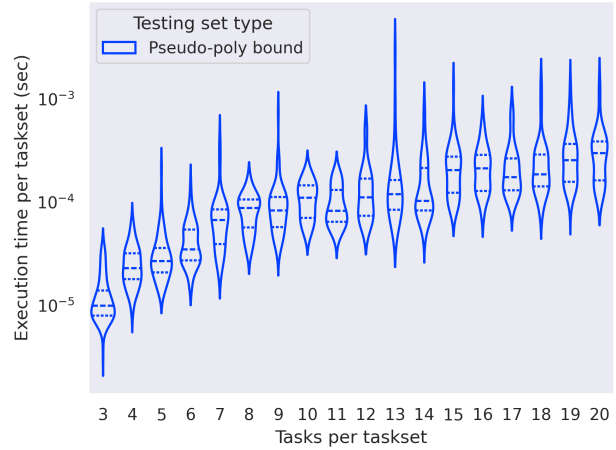
677 improvements enable testing larger task sets, as shown in the following sections.

678 In this work, we also introduce an optimization for implicit-deadline task systems that allows  
 679 us to avoid testing timepoints past  $D_{\max}$ . In Figure 8, we analyze the impact of this optimization.  
 680 Using the full, exponential testing set from the prior work, our implementation is able to test  
 681 tasksets with 6 tasks in only a few seconds, but the time needed still scales exponentially with the  
 682 number of tasks; past eight tasks per set, the analysis once again takes an unreasonable amount  
 683 of time to run. Using the optimization for implicit-deadline systems, evaluating schedulability is  
 684 orders of magnitude faster, and the evaluation times also become more consistent for each taskset.  
 685 This optimization allows us to determine the schedulability ratio for systems of up to 20 tasks,  
 686 the results of which are displayed in Figure 3. As the time needed to determine schedulability  
 687 now scales much more slowly with respect to the number of tasks, we believe that evaluating even  
 688 larger task sets is also possible.

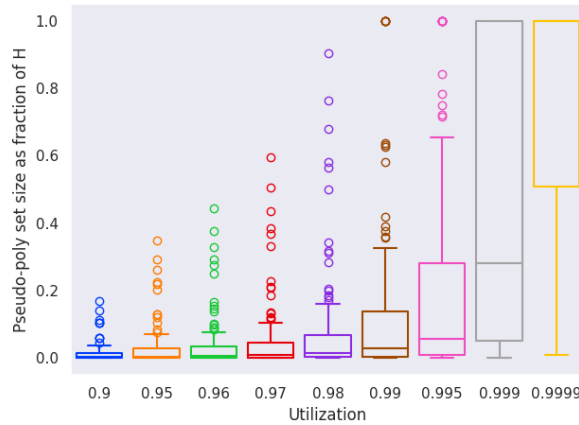
689 In Figure 9, we evaluate the performance of determining schedulability of constrained-deadline



■ **Figure 8** Time to test schedulability of sets of implicit-deadline tasks of varying sizes with periods of 10–30 time units, 1–4 phases, and utilization of 0.9, either testing up to the hyperperiod or stopping at  $D_{\max}$ . Note the logarithmic scale.



■ **Figure 9** Time to test schedulability of sets of constrained-deadline tasks of varying sizes with periods of 10–30 time units, 1–4 phases, and utilization of 0.9, using the pseudo-polynomial testing set bound from [6]. Note the logarithmic scale.



■ **Figure 10** 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.

690 task sets using the pseudo-polynomial bound on the testing set identified in [6]. Compared to the  
 691 full hyperperiod testing set, running the evaluation with this testing set is much faster. However,  
 692 we note that it is still slower than the optimized implicit-deadline bound from Figure 8. Moreover,  
 693 the distribution of execution times exhibits high variability. Perhaps most importantly, our  
 694 evaluation is on task sets where  $U = 0.9$ , at which the pseudo-polynomial bound is still reasonably  
 695 small. As shown in Figure 10, when  $U \rightarrow 1$ , the size of the pseudo-polynomial set approaches the  
 696 hyperperiod set, and it becomes increasingly less feasible to iterate over the entire testing set.

## 697 **7** Related Work

698 The trade-off between security and timing constraints in real-time, IoT, and edge computing  
 699 environments is crucial as it involves balancing robust security measures with the need for real-time  
 700 performance. Several studies have explored these trade-offs. For instance, Leonardi et al. [25] and  
 701 Lemieux-Mack et al [24] present mixed-integer linear programming approaches to optimize security  
 702 and schedulability in real-time embedded systems under cyber-attacks. Wang et al. [41] consider  
 703 the trade-off between security and overhead for pointer integrity checks. These papers attempt to  
 704 maximize security without overloading the target system, but do not consider preemption-related  
 705 overhead induced by the startup/teardown costs of the selected security mechanisms.

706 Limited-preemption scheduling, surveyed in 2013 by Buttazzo et al. [16], balances the increased  
 707 blocking times of non-preemptive execution with the increased overheads caused by context  
 708 switching. The original models of Baruah and Bertogna minimize context switching by finding the  
 709 longest time each task may execute non-preemptively without compromising schedulability [7, 12],  
 710 but do not account for the worst-case overhead due to the resulting preemptions.

711 Later approaches account for both blocking time and preemption overheads by defining  
 712 instants that preemption can occur during task execution (called *preemption points*) to guarantee  
 713 schedulability. In [13], Bertogna et al. develop an algorithm to find preemption points for tasks  
 714 scheduled with EDF under the assumption that the preemption overhead for each task is constant.  
 715 In contrast, in our MPS task model, individual task phases have unique preemption costs. A  
 716 similar model to that of [13], but for fixed-priority (FP) scheduling, again with constant preemption  
 717 overheads for each task, was developed by Yao et al. in [44].

718 In [14], Bertogna et al. model tasks as sequences of “basic blocks,” and present algorithms for  
 719 selecting preemption points between those blocks for both EDF and FP scheduling. Although  
 720 less flexible than the earlier “floating” preemption point models where a preemption point can  
 721 be placed anywhere, that model allows different preemption costs between blocks. The latter  
 722 is similar to the algorithm in [13], but supports domain-specific preemption costs rather than a  
 723 constant overhead per task.

724 Other work on limited-preemption scheduling, including cache-aware analysis [31], probabilistic  
 725 [30] or “typical” execution models [9] are outside the scope of this paper.

## 726 **8** Conclusions

727 We believe that the concurrent consideration of timing and security properties within a single unified  
 728 framework is an effective means of extending the rigorous approach of real-time scheduling theory  
 729 to guaranteeing appropriately-articulated security properties in resource-constrained embedded  
 730 systems. In real-time scheduling theory, pre run-time verification of timing correctness is performed  
 731 using models of run-time behavior; these models are carefully crafted for specific purposes: e.g,  
 732 the sporadic task model [6] has been designed to represent recurrent processes for which it is safe  
 733 to assume a minimum duration between successive invocations and for which timing correctness is  
 734 defined as the ability to meet all deadlines.

735 In this work, as in [8], we have extended the sporadic task model in a security-cognizant  
 736 manner, to deal with a particular kind of security-cognizant workload model. For the specific  
 737 model that we have proposed, we have developed algorithms that are able to provide provable  
 738 correctness of both the timing and the security properties that are considered.

739 In this extension to our original work in [8], we have corrected the long-standing condition  
 740 of Baruah and Bertogna [7, 12] for EDF schedulability of limited-preemption tasks. Leveraging  
 741 this, we reduced the execution time complexity of our algorithm, and demonstrated that in an  
 742 implicit-deadline task system, it can run quickly even for large numbers of tasks.



We note that the improvements made to the algorithm in this extension also apply to the generalization of the model to conditional code that we presented in [8]. In an already-published extension to that work [39], we further improve on the conditional execution model by removing the assumption that startup/teardown costs are paid at every phase transition and analyze its performance.

The complexity improvements made in this extension, in combination with a more efficient implementation of the algorithm, 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 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 OP-TEE. URL: <https://www.trustedfirmware.org/projects/op-tee/>.
- 2 OP-TEE Documentation: Core: Pager. URL: <https://optee.readthedocs.io/en/latest/architecture/core.html#pager>.
- 3 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.
- 4 Mohammad Fakhruddin Babar and Monowar Hasan. Deeptrust<sup>rt</sup>: Confidential deep neural inference meets real-time! In *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2024.
- 5 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.
- 6 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.
- 7 Sanjoy Baruah. The limited-preemption uniprocessor scheduling of sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, 2005.
- 8 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.
- 9 Sanjoy Baruah and Nathan Fisher. Choosing preemption points to minimize typical running times. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems, RTNS '19*, page 198–208, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3356401.3356407.
- 10 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.
- 11 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.
- 12 Marko Bertogna and Sanjoy Baruah. Limited preemption EDF scheduling of sporadic task systems. *IEEE Transactions on Industrial Informatics*, 2010.
- 13 Marko Bertogna, Giorgio Buttazzo, Mauro Marinoni, Gang Yao, Francesco Esposito, and Marco Caccamo. Preemption Points Placement for Sporadic Task Sets. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 251–260, 2010. doi:10.1109/ECRTS.2010.9.

- 14 Marko Bertogna, Orges Xhani, Mauro Marinoni, Francesco Esposito, and Giorgio Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *2011 23rd Euromicro Conference on Real-Time Systems*, pages 217–227. IEEE, 2011.
- 15 Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1-2), 2005.
- 16 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.
- 17 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.
- 18 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.
- 19 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.
- 20 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.
- 21 Pontus Ekberg. *Models and Complexity Results in Real-Time Scheduling Theory*. PhD thesis, Ph.D. thesis, Uppsala University, 2015.
- 22 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.
- 23 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.
- 24 Cailani Lemieux-Mack, Kevin Leach, Ning Zhang, Sanjoy Baruah, and Bryan C Ward. Optimizing runtime security in real-time embedded systems. In *Proc. of Workshop on Optimization for Embedded and Real-time Systems (OPERA)*, December 2024.
- 25 Sandro Di Leonardi, Federico Aromolo, Pietro Fara, Gabriele Serra, Daniel Casini, Alessandro Biondi, and Giorgio Buttazzo. Maximizing the security level of real-time software while preserving temporal constraints. *IEEE Access*, 11:35591–35607, 2023. URL: <http://dx.doi.org/10.1109/ACCESS.2023.3264671>, doi:10.1109/access.2023.3264671.
- 26 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.
- 27 C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
- 28 Yin Liu, Siddharth Dhar, and Eli Tilevich. Only pay for what you need: Detecting and removing unnecessary TEE-based code. *Journal of Systems and Software*, 188:111253, 2022. Publisher: Elsevier.
- 29 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.
- 30 Filip Markovic, Jan Carlson, Radu Dobrin, Bjorn Lisper, and Abhilash Thekkilakattil. Probabilistic response time analysis for fixed preemption point selection. In *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10, 2018. doi:10.1109/SIES.2018.8442099.
- 31 Filip Marković, Jan Carlson, and Radu Dobrin. Cache-aware response time analysis for real-time tasks with fixed preemption points. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 30–42, 2020. doi:10.1109/RTAS48715.2020.00–19.
- 32 Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for TCB minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 315–328, 2008.
- 33 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.
- 34 Sabin 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.
- 35 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.
- 36 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.
- 37 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.
- 38 M. Pajic, N. Bezzo, J. Weimer, R. Alur, R. Mangharam, N. Michael, G.J. Pappas, O. Sokol-

- sky, 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.
- 39 Fatima Raadia, Nathan Fisher, Thidapat Chantem, and Sanjoy Baruah. An improved security-cognizant scheduling model. In *2024 IEEE 27th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 1–8. IEEE, 2024.
- 40 Jinwen Wang, Ao Li, Haoran Li, Chenyang Lu, and Ning Zhang. Rt-tee: Real-time system availability for cyber-physical systems using arm trustzone. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 352–369. IEEE, 2022.
- 41 Yujie Wang, Cailani Lemieux-Mack, Thidapat Chantem, Sanjoy Baruah, Ning Zhang, and Bryan C Ward. Partial context-sensitive pointer integrity for real-time embedded systems. In *2024 IEEE Real-Time Systems Symposium (RTSS)*, pages 415–426. IEEE Computer Society, 2024.
- 42 T. Xie and X. Qin. Scheduling security-critical real-time applications on clusters. *IEEE Trans. Computers*, 55(7):864–879, July 2006.
- 43 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.
- 44 Gang Yao, Giorgio Buttazzo, and Marko Bertogna. Feasibility analysis under fixed priority scheduling with limited preemptions. *Real-Time Systems*, 47(3):198–223, 2011. Publisher: Springer.
- 45 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.
- 46 Man-Ki Yoon, Sabin 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.