In Search of Butterflies: Exceedance Analysis for Real-Time Systems under Transient Overload

Matteo Zini[1] Filip Marković[2] Daniel Casini[1,3] Alessandro Biondi[1,3] Björn B. Brandenburg[2]
[1]*TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy*
[2]*Max Planck Institute for Software Systems, Kaiserslautern, Germany*
[3]*Department of Excellence in Robotics & AI, Scuola Superiore Sant'Anna, Pisa, Italy*

***Abstract*—In theory, real-time systems are provisioned based on provably sound *worst-case execution times* (WCETs), but in practice often only empirically derived, unsound execution-time estimates—*i.e.*, *nominal execution times* (NETs)—are available since WCETs are difficult to obtain on modern hardware. NETs pose two significant challenges: First, since NETs may be exceeded at runtime, any response-time bounds derived from NETs are transitively unsound and may be violated. Second, even a minuscule NET violation can result in large, *nonlinear* response-time increases due to hard-to-predict, cascading scheduling effects. To explore the risk NET exceedance poses to a system's temporal correctness, this paper provides the first *general*, *systematic*, and *explainable* methodology for *exceedance analysis*. The proposed approach supports fixed-priority (FP), earliest-deadline first (EDF), and first-in first-out (FIFO) scheduling on a uniprocessor or within a partitioned multiprocessor platform, and the full spectrum of preemption models from fully preemptive to fully non-preemptive workloads. Additionally, it produces explainable evidence in the form of tunable example traces that engineers can adjust to take system-specific expertise into account. The proposed methodology is evaluated with synthetic task sets and workloads based on an automotive benchmark, and in a case study applied to parts of the WATERS'17 industrial challenge.**

I. Introduction

The temporal correctness of a real-time system fundamentally depends on the maximum resource requirements of the tasks comprising the system. Traditionally, this aspect is captured by the concept of a task's *worst-case execution time* (WCET), which is famously one of the key components of schedulability analysis. Ideally, each task's WCET should be upper-bounded through sound static analysis [*e.g.*, 38, 70], thereby ensuring the system can meet its timing constraints under all conditions.

In practice, however, modern real-time systems are often deployed on hardware and software stacks that are beyond the reach of state-of-the-art WCET analysis. For instance, real-time Linux variants, commonly deployed on temporally unpredictable hardware, are used in time-sensitive applications such as *unmanned aerial vehicles* (UAVs) [*e.g.*, 24, 32, 48], *autonomous driving* [*e.g.*, 42, 43, 67], and *spacecraft* [52].

On such platforms, few viable options remain, with rigorous testing and measurement-based approaches as the only practical solutions applicable today. These methods, enhanced by the integration of safety margins, yield execution-time "bounds" that, while not provably sound, are deemed sufficiently reliable in many application domains. In the following, we refer to such estimated execution-time bounds as *nominal execution times* (NETs). A typical use of NETs is as a proxy for (unknown) WCET parameters in response-time analysis.

The resulting *nominal response-time bounds* (NRTs), computed based on empirically approximated model parameters that may be optimistic, are clearly not provably sound guarantees. Nonetheless, NRTs are arguably better than nothing in practice and thus commonly used [8, Q18 & Q23]. In fact, NRTs can appear so reliable during testing that they may lull practitioners into a false sense of safety, for the following reasons:

1) In many, or even most, real-world systems, not every task operates under a strict deadline. Thus, minor response-time increases beyond nominal bounds are tolerable and may not even be logged or noticed for all tasks.
2) Critical time-sensitive tasks (*i.e.*, those with strict deadlines) typically have some (static) slack [34], with NRTs well below their actual deadlines, often so by a considerable margin. This slack provides a safety margin before an NRT violation becomes a significant problem.
3) In most cases, the impact of NET exceedance is only *linear*: a slight NET exceedance leads to only a proportionally small increase in NRT. Therefore, considering points (1) and (2), the effects of a few processor cycles of exceedance "here or there" are usually negligible.

Consequently, it can be tempting to believe that NRTs are generally "in the right ballpark" and that minor NET fluctuations cause only negligible "response-time noise." That thinking, alas, is flawed and potentially dangerous: in pathological situations, even a minuscule NET exceedance can lead to a much larger, *nonlinear* increase in the response time of *some* task (not necessarily the one exceeding its NET) [20, 30].

If a deployed system encounters such a nonlinear increase, its behavior can drastically and very suddenly diverge from its "good" behavior observed during testing, resulting in massive and likely unanticipated timing violations. For example, in systems featuring even partial non-preemptive execution, NET exceedance can trigger timing anomalies such as the *self-pushing phenomenon* [18]. A single case of NET exceedance can result in a *domino effect*, leading to *progressively longer* response times that may cascade across multiple jobs.

Unfortunately, the pathological scenarios that result in nonlinear increases are neither obvious nor unique and do not follow a simple-to-predict pattern (as illustrated in Sec. II). Furthermore, even if the first encountered nonlinearity is tolerable, the next nonlinear step that pushes the system over the proverbial edge (*e.g.*, past a hard deadline) may lurk nearby. Therefore, when working with NETs—which is to say, for a vast number of systems in practice—it is essential to understand the space of possible *NET exceedance effects*. In particular, it is crucial to assess the following two questions.

1) How close is the system to its first nonlinear response-time increase, *i.e.*, is there a comfortably large engineering margin before exceedance effects become hard to predict?
2) Are there additional nonlinearities "hiding" beyond the first one, and what is their potential impact?

Neither question is easy to answer in general. Moreover, brute-force search proves impractical due to the vast combinatorial space of potential NET exceedance scenarios. Readers familiar with classic sensitivity analysis (*e.g.*, [49]) will recognize a similarity in motivation, but as reviewed in Sec. IX, existing sensitivity analyses cannot fully answer the above questions due to differences in assumptions and approach.

This paper. We provide the first *general*, *systematic*, and *explainable* methodology for exploring a given workload's space of nonlinear response-time increases. Our approach applies to workloads on a uniprocessor or within a partitioned multiprocessor platform. Foremost, it is *general* as it works with the *fixed-priority* (FP), *earliest-deadline first* (EDF), and *first-in first-out* (FIFO) scheduling policies. It also supports the general limited-preemptive scheduling model [19] that spans the full spectrum of task preemption from *fully preemptive* to *non-preemptive*, thus covering most scheduling approaches encountered in practice. Additionally, the proposed methodology is *systematic* in that it uses a novel algorithm that enumerates all nonlinear steps, thereby allowing the sparse space of response-time nonlinearities to be explored without a brute-force search. Last but not least, we propose a flexible trace generator that, for any identified nonlinearity, produces concrete example scenarios triggering the identified nonlinear response-time increase. Such traces can be readily understood by practitioners without a background in scheduling theory. Thus, our method aligns with the growing demand for *explainability* in real-time computing [*e.g.*, 9, 12, 54]. Notably, the trace generator provides intuitive *tuning knobs* that allow engineers to incorporate application-specific knowledge (*e.g.*, differences in reliability attributed to each task's NET). In summary:

- We generalize existing RTAs for FP, EDF, and FIFO scheduling to explicitly account for the cumulative effects of NET exceedance by any number of tasks (Sec. IV).
- We provide an algorithm for interactive exploration of the space of response-time nonlinearities. (Sec. V).
- We state the problem of finding illustrative example traces as an optimization problem, and propose configuration parameters to integrate system-specific expertise (Sec. VI).
- We evaluate the exploration algorithm's efficiency and the trace generator's capabilities with synthetic task sets and workloads from an automotive benchmark [44] (Sec. VII).
- Finally, we report on a case study involving a workload extracted from the WATERS'17 challenge [35] to illustrate the practical utility of exceedance analysis (Sec. VIII).

TABLE I: Example Task Set

Task	Priority	Period	Deadline	NET (per segment)
τ_1	3	50 ms	50 ms	$\langle 12\,\text{ms}\rangle$
τ_2	2	80 ms	80 ms	$\langle 30\,\text{ms}\rangle$
τ_3	1	200 ms	200 ms	$\langle 26\,\text{ms}, 25\,\text{ms}, 10\,\text{ms}\rangle$

II. Motivating Example

The limited-preemptive task set given in Table I illustrates the main problem and key concepts central to this paper. For each of a task's segments, a per-segment NET is given, *e.g.*, the maximum execution time observed during testing. Tasks τ_1 and τ_2 consist of one non-preemptive segment each with NETs of 12 ms and 30 ms, respectively. Task τ_3 comprises three non-preemptive segments with NETs of 26 ms, 25 ms, and 10 ms, respectively. Fig. 1 depicts three possible schedules of the workload under limited-preemptive FP scheduling. Task τ_1 has the highest priority; τ_3 the lowest.

Nominal execution. Fig. 1(a) shows a nominal scheduling scenario, wherein all jobs exhibit execution costs matching exactly their corresponding tasks' NETs. Here, the NRT of task τ_3 is 157 ms. That is, a response-time analysis using the parameters in Table I arrives at 157 ms as τ_3's claimed response-time bound, as illustrated in Fig. 1(a).

Linear increase. Fig. 1(b) illustrates the effect of "benign" NET exceedance. Here, the second job of τ_1 exceeds its NET by 1 ms, thus executing in total for $12\,\text{ms} + 1\,\text{ms}$. As the NET was empirically derived, is not too surprising that it can be exceeded once in a while by a small amount. Since τ_1's execution delays the execution of τ_3, τ_1's NET exceedance affects τ_3's response time, which increases linearly (w.r.t. the total amount of exceedance) to 158 ms.

Now, if the first job of task τ_2 were to *simultaneously* exceed its NET by 1 ms (for a total of 30 ms+1 ms), τ_3 would suffer the combined delay due to both overruns. However, the cumulative effect would still be linear in the total exceedance: τ_3's response time would increase to 159 ms, exceeding its NRT by 2 ms. Even then there would still be 41 ms of slack remaining before τ_3 misses its deadline (at time 200 ms), which may *seem* like a reassuring safety margin. However, it is not.

Nonlinear step. Fig. 1(c) illustrates the central issue addressed in this paper: priority-based scheduling is subject to *nonlinearities* akin to the infamous *Butterfly effect*, wherein a tiny change in initial conditions can result in catastrophically different outcomes. Continuing the example, suppose that the first segment of τ_3 itself also exceeds its NET by 1 ms, for a total of 3 ms of exceedance across all three tasks. Unlike in the previous case, τ_3's response increases from the nominal 157 ms to 202 ms, resulting in a deadline miss. That is, going from 2 ms of total exceedance to 3 ms consumes the *entire* 41 ms of slack observed in the previous case.

The example demonstrates that real-time scheduling with uncertain parameters (NETs) is subject to dangerous nonlinearities that arise from complex interactions among multiple tasks. In general, it is hard to predict where such nonlinearities lurk in the "exceedance space" because they are sparse and do not

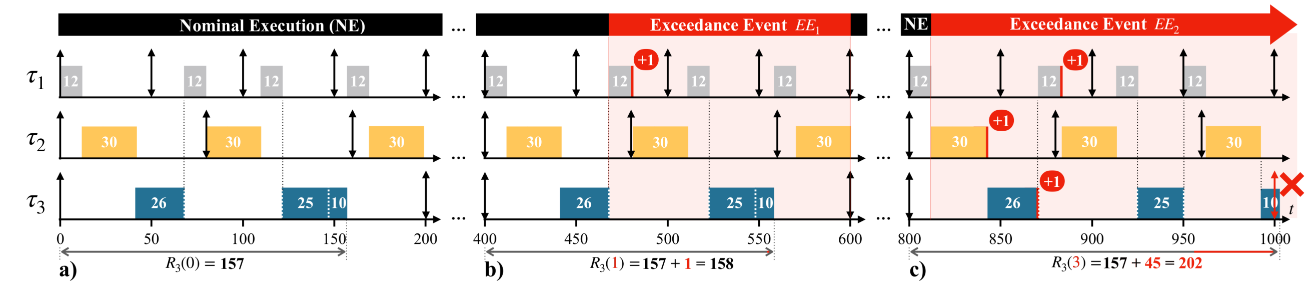

Fig. 1: Example execution traces of the periodic tasks given in Table I, illustrating the response time of τ_3 in three cases: **(a)** all jobs exhibit nominal execution times; **(b)** a job of τ_1 exceeds its NET by 1 ms; and **(c)** a job of each task exceeds its NET by 1 ms. *Legend*: ↑ arrival of a job, ↓ absolute deadline of a job, ↕ coinciding arrival and deadline.

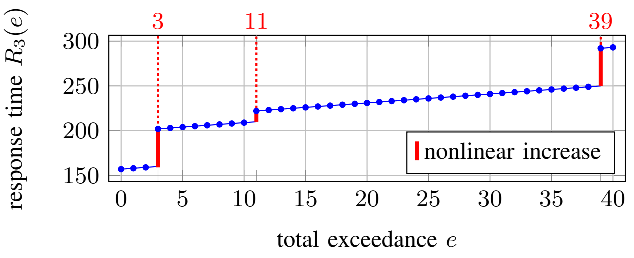

Fig. 2: Task τ_3's response time $R_3(e)$ *vs.* total exceedance e.

follow a simple pattern. Case in point, Fig. 2 charts the response time of τ_3 as a function of the total NET exceedance of all tasks. While *most* NET exceedance increases have only linear impact, a disproportionally large increase in τ_3's response time occurs when total exceedance reaches 3 ms, 11 ms, or 39 ms. These are the scenarios engineers should pay special attention to.

Just relying on NRTs and simple slack metrics is clearly not enough to characterize a system's true resilience to NET overruns. However, the only existing alternative, classic sensitivity analysis (Sec. IX), cannot identify the specific nonlinearity scenarios in question, and does not support (partially) non-preemptive workloads. There is thus a need for a systematic and general approach to understanding *how far away* a system is from any "surprises" (*i.e.*, nonlinearities) in the exceedance space. In this paper, we propose the first such method.

III. System Model and Notation

We assume a discrete time model and let $\varepsilon \triangleq 1$ denote the least indivisible time unit (*e.g.*, one processor cycle).

We consider a set of n sporadic tasks $\tau = \{\tau_1, ..., \tau_n\}$. Task τ_i's NET is denoted by C_i. Formally, we make no assumption on how C_i is derived nor on its relationship to τ_i's true WCET, which we assume to be unknown. In practice, C_i is likely derived from the *maximum observed execution time*, possibly with an additional safety margin, but engineers may choose to set C_i to a value even somewhat below the observed maximum (*e.g.*, the 95th percentile of the observed execution-time distribution in a soft real-time system), reflecting the designer's tolerance for risk and aversion to over-provisioning.

We denote the j-th activation (or *job*) of task τ_i as $J_{i,j}$, and denote its *arrival time*, *execution time*, and *exceedance* as $a_{i,j}$, $c_{i,j}$, and $e_{i,j} = \max(0, c_{i,j} - C_i)$, respectively. We omit the job index j and simply write J_i when the job index is irrelevant or clear from context. Job arrivals are constrained by the task's *arrival curve* $\alpha_i(\Delta)$, which bounds the number of jobs of τ_i arriving in any interval of length Δ, such that $\forall t \in \mathbb{N}, \forall \Delta \in \mathbb{N},\ |\{J_{i,j} : t \le a_{i,j} < t + \Delta\}| \le \alpha_i(\Delta)$. This generalizes both the classic periodic and sporadic task models: the arrival curve of a periodic task τ_i with *maximum release jitter* ι_i and *period* T_i is $\alpha_i(\Delta) = \lceil(\Delta + \iota_i)/T_i\rceil$, and the arrival curve of a sporadic task τ_i with *minimum inter-arrival time* MIT_i is $\alpha_i(\Delta) = \lceil \Delta / \mathrm{MIT}_i \rceil$.

Task τ_i's (nominal) *request-bound function* is $RBF_i(\Delta) \triangleq \alpha_i(\Delta) \cdot C_i$. We assume the system is not running at full utilization if all jobs execute according to their NETs (*i.e.*, $\exists \Delta, \sum_{\tau_i \in \tau} RBF_i(\Delta) < \Delta$), and thus it can eventually recover from finite transient overruns (*i.e.*, NET exceedances).

We assume the workload τ executes on a unit-speed uniprocessor, or equivalently, on a processor that is part of a partitioned multiprocessor system. Overheads such as context-switching costs are not modeled explicitly but implicitly accounted for as part of NETs and job execution times.

In this paper, we focus on the well-known FP, EDF, and FIFO scheduling policies, which are work-conserving. In the context of EDF, we let D_i denote task τ_i's *relative deadline*, which determines a job $J_{i,j}$'s *absolute deadline* (and hence job priority) as $a_{i,j} + D_i$. Under FP scheduling, π_i denotes the *priority* assigned to task τ_i, where $\pi_i > \pi_h$ indicates that τ_i has higher priority than τ_h. Under FIFO scheduling, jobs are executed in the order of their arrival, without considering task priority or deadline. A job's arrival time can be seen as its priority, with earlier arrivals having higher priority.

We consider four commonly encountered preemption models [19]. If tasks are *fully preemptive*, jobs can be interrupted at any point during their execution; conversely, jobs of *fully non-preemptive* tasks must run to completion once they have started execution. As a middle ground, jobs of *segmented limited-preemptive* tasks permit preemption only at specific *preemption points* that form the boundaries between non-preemptive segments. We let $C_{i,k}$ denote the NET of τ_i's kth non-preemptive segment, and m_i the number of non-preemptive segments of τ_i. Finally, under the *floating non-preemptive* model, jobs execute non-preemptive sections at *a priori* unknown times, but for each task τ_i a bound γ_i on the NET of any of its non-preemptive sections is given.

As summarized in Table II, all four preemption models can be

TABLE II: Considered Preemption Models

Preemption Model	RCT_i	NPS_i
Fully preemptive	C_i	ε
Fully non-preemptive	ε	C_i
Segmented non-preemptive	$C_i - (C_{i,m_i} - \varepsilon)$	$\max_k\{C_{i,k}\}$
Floating non-preemptive	C_i	γ_i

captured with two parameters: the *run-to-completion threshold* RCT_i bounds the amount of execution any job of task τ_i must complete before it is guaranteed to finish execution without further preemptions, and the *longest non-preemptive section* NPS_i bounds the maximum time for which a job of τ_i executes non-preemptively. As is the case with C_i, these parameters are *nominal* bounds that may be exceeded at runtime.

IV. Exceedance-Aware Response-Time Analysis

To understand the effects of exceedance, we require a bound on the response time of a task during an *exceedance event* (EE), *i.e.*, in an interval in which one or more jobs exceed their NETs. More precisely, if the *total exceedance* of all jobs during an EE is at most e time units, we seek to bound the maximum response time $R_i(e)$ for each $\tau_i \in \tau$, taking into account that the total exceedance e may be spread arbitrarily across any combination of jobs and tasks. For example, Fig. 2 shows $R_3(e)$.

A. Exceedance Events

An EE starts when any job exceeds its NET and lasts until all transient effects caused by exceedance have fully dissipated. EEs are inherently disjoint: if some job exceeds its NET while the system is still recovering from the exceedance imposed by an earlier job, then these two jobs are by definition part of the same EE. We place restrictions on neither the maximum number of exceeding jobs involved in an EE, nor its length. For example, Fig. 1(b) shows an EE involving one job spanning from $t = 468$ to $t = 600$, and Fig. 1(c) shows an EE involving three jobs spanning from $t = 812$ to $t = 1200$ (not shown).

As the system is not running at full utilization (with regard to NETs), in the worst case, an EE ends when the system becomes idle. In practice, exceedance may dissipate sooner (*i.e.*, cease to alter the nominal schedule) if exceedance in one job happens to be compensated for by another job under-running its NET.

Clearly, for $e = 0$ (*i.e.*, without exceedance), the problem of bounding $R_i(e)$ reduces to a regular *response-time analysis* (RTA). We hence build on existing RTAs, which we review next.

B. The Baseline RTAs

We build on a family of RTAs derived from Bozhko and Brandenburg's *abstract RTA* (aRTA) framework [16]. Both aRTA and its concrete instantiations for the scheduling policies and preemption models considered in this paper have been formally verified [13, 16] within the Prosa project [4, 22] using the Coq proof assistant [2]. In the following, we briefly review the *intuition* underlying aRTA and refer the interested reader to the original papers [13, 16] for a more formal derivation.

aRTA is based on the classic *busy-window principle* [50], which builds upon two key concepts: *quiet times* and the *busy window*. A time instant t is a *quiet time* with respect to a job J_i if all jobs of priority higher than or equal to J_i released *prior* to time t have completed *by* time t (*i.e.*, there is no *carry-in work* at time t) [16]. Then an interval $[t_1, t_2)$ is J_i's *busy window* if and only if t_1 is the only quiet time in $[t_1, t_2)$, t_2 is a quiet time, and J_i is released in the interval. A job's busy window, if it exists, is unique and ensures that the job is finished by the end of the window. A job may not have a busy window if the system is *permanently* overloaded.

TABLE III: $\mathbb{HEP}(i)$ and $\mathbb{B}(i, A)$ for FP, EDF, and FIFO.

Policy	$\mathbb{HEP}(i)$	$\mathbb{B}(i, A)$
FP	$\{\tau_h \in \tau \mid \tau_h \neq \tau_i, \pi_i \leq \pi_h\}$	$\{\tau_h \in \tau \mid \pi_i > \pi_h\}$
EDF	$\tau \setminus \{\tau_i\}$	$\{\tau_h \in \tau \mid D_h > D_i + A\}$
FIFO	$\tau \setminus \{\tau_i\}$	$\emptyset$

As a notational convention, we let τ_i denote the task under analysis, $J_{i,j}$ (or simply J_i) an arbitrary job of τ_i, and A the relative *arrival offset* of $J_{i,j}$ in its busy window (*i.e.*, if $[t_1, t_2)$ is $J_{i,j}$'s busy window, then $A = a_{i,j} - t_1$).

Additionally, to generalize over the considered scheduling policies and preemption models, we let $\mathbb{HEP}(i)$ denote the set of tasks other than τ_i that can potentially release *higher-or-equal-priority* jobs with respect to τ_i, define $\mathbb{HEP}^+(i) = \mathbb{HEP}(i) \cup \{\tau_i\}$, and $\mathbb{B}(i, A)$ as the set of tasks that can release jobs causing *priority-inversion blocking* to a job of τ_i released with offset A. Table III gives concrete definitions of $\mathbb{HEP}(i)$ and $\mathbb{B}(i, A)$ for the three considered scheduling policies.

Given these key definitions and a task under analysis τ_i for which we seek to obtain a response-time bound, each of the considered RTAs consists of the following steps: **(i)** obtain a bound L_i on the duration of the *longest-possible busy window* of any job of τ_i, **(ii)** define a sparse search space $\mathcal{A}_i$ of relevant *relative arrival offsets*, and **(iii)** solve a policy-specific *response-time recurrence* for each $A \in \mathcal{A}_i$ to identify the final bound.

Regarding step (i), Table IV defines safe (but not necessarily tight) upper bounds on L_i for the considered policies [4]. (The blocking bound $B_i(0)$ is defined in Table V.) If no such L_i can be found (*e.g.*, via fixed-point search), the system is potentially overloaded and no finite response-time bound can be found.

The second step is to compute the set $\mathcal{A}_i$ of all arrival offsets that must be considered to obtain a correct response-time bound. In general, it is not obvious *a priori* which release offset A results in the maximum response time. However, it has been proven [16] that the arrival offset that generates the maximum response time belongs to the set $\mathcal{A}_i \triangleq \{0\} \cup \{0 < A < L_i \mid \exists\Delta,\ IBF_i(A - \varepsilon, \Delta) \neq IBF_i(A, \Delta)\}$, where $IBF_i(A, \Delta)$ is a policy-specific *interference bound function*. More specifically, $IBF_i(A, \Delta)$ upper-bounds the maximum total interference that any job J_i of task τ_i can experience in any interval of length Δ, assuming J_i arrives A time units after the beginning of its busy window. Table V defines $IBF_i(A, \Delta)$ for the considered scheduling policies [4, 13, 16]. Simply checking each offset in the search space $\mathcal{A}_i$ is sufficient to bound task τ_i's maximum response time. Note that $\mathcal{A}_i$ can be efficiently enumerated for each of the considered policies by focusing only on the "steps" of the underlying arrival curves [4, 54].

TABLE IV: Bounds on L_i for FP, EDF [4] and FIFO [13].

Policy	L_i
FP	$\min\{L > 0 \mid L \geq B_i(0) + \sum_{h \in \mathbb{HEP}^+(i)} RBF_h(L)\}$
EDF	$\min\{L > 0 \mid L \geq \sum_{\tau_h \in \tau} RBF_h(L)\}$
FIFO	$\min\{L > 0 \mid L \geq \sum_{\tau_h \in \tau} RBF_h(L)\}$

Step (iii) is to find the following fixed point for each $A \in \mathcal{A}_i$:

$$\min_{F_i^A > 0}\{A + F_i^A = RCT_i + IBF_i(A, A + F_i^A)\}. \qquad (1)$$

Intuitively, the term F_i^A in Eq. (1) upper-bounds the response time of the *preemptive part* of J_i (*i.e.*, before reaching the final non-preemptive part characterized by RCT_i), under the assumption that the job has the arrival offset A.

Lastly, suppose that a fixed-point solution F_i^A is found for each $A \in \mathcal{A}_i$, and let $F_i = \max_{A \in \mathcal{A}_i}\{F_i^A\}$. The overall response-time bound is then $R_i = F_i + (C_i - RCT_i)$. If no solution F_i^A can be found for some A, or if no bound L_i can be found, then no response-time bound can be established and R_i is undefined (*i.e.*, the analysis procedure returns an error).

C. Augmenting RTAs for Exceedance Awareness

We now come to our first main contribution, which is the integration of exceedance into the RTAs just summarized. More precisely, we seek to upper-bound the response time of a task under analysis τ_i *during an EE* in which arbitrary jobs collectively exhibit a total exceedance of at most e time units. Of course, we generally do not *a priori* know the total exceedance e, but momentarily postpone the problem of finding "interesting" values of e to consider and focus here solely on the problem of bounding $R_i(e)$ for any given, fixed $e \geq 0$.

Recall that we focus on systems that have been designed to be resilient to *transient* overloads by leaving a *safety margin* with respect to nominal utilization (*i.e.*, the system is not fully utilized outside of EEs). Under this assumption, a finite amount of exceedance e causes only *transient* overload and not a potentially permanent, unrecoverable divergence from nominal execution behavior. It is thus possible to provide meaningful response-time guarantees during an EE.

To this end, it is necessary to: **(1)** derive an *exceedance-aware bound* $L_i(e)$ on the maximum length of a busy window during an EE; **(2)** augment the space of relevant arrival offsets $\mathcal{A}_i(e)$; **(3)** for every $A \in \mathcal{A}_i(e)$, compute an exceedance-aware, per-offset bound $F_i^A(e)$, considering that exceedance can manifest in both higher- and lower-priority jobs; and **(4)** find the *exceedance-aware response-time bound* $R_i(e) = F_i(e) + (C_i - RCT_i)$, where $F_i(e) \triangleq \max_{A \in \mathcal{A}_i}\{F_i^A(e)\}$. As we will see shortly, steps (2) and (4) are straightforward, while steps (1) and (3) require some further analysis.

1) Maximum busy-window length: Let us first consider bounding $L_i(e)$ under EDF and FIFO, which in the case without exceedance have simpler bounds than FP (recall Table IV).

Exceedance can generally occur in any task and any job, at various times in a busy window. To simplify the analysis, we can *reinterpret* a given schedule *with* exceedance as a schedule *without* exceedance by introducing—purely as a modeling construct—a hypothetical *exceedance task* τ_E. We then can attribute to (hypothetical) jobs of τ_E any execution time that goes beyond a (regular) job's nominal execution-time bound.

To account for the effect of exceedance on the maximum busy-window length, we can then treat the exceedance task τ_E as a regular task when computing the bound, which hence must satisfy $L \geq RBF_E(L) + \sum_{\tau_h \in \tau} RBF_h(L)$ for some $L > 0$. Additionally, by initial assumption, all the (hypothetical) jobs of task τ_E contributing to the busy window collectively consume exactly e time units. Therefore, $RBF_E(L) = e$. We thus arrive at the following exceedance-aware bound on the maximum busy-window length under EDF and FIFO:

$$L_i(e) \triangleq \min\{L > 0 \mid L \geq e + \textstyle\sum_{\tau_h \in \tau} RBF_h(L)\}.$$

Let us now turn to FP scheduling, where we must consider that exceedance can occur both in jobs of lower priority and in jobs of higher or equal priority with respect to J_i. Therefore, we split the total exceedance in two parts $e^{\text{hep}} + e^{\text{lp}} = e$, where e^{hep} accounts for exceedance in high-priority jobs, and likewise e^{lp} for exceedance in lower-priority jobs.

The exceedance produced by lower-priority jobs e^{lp} is easy to analyze since, in the worst case, it manifests completely in the non-preemptive segment causing J_i to incur priority inversion (at the beginning of its busy window [16]). We thus account for it simply by including it in the blocking bound: $B_i'(A, e^{\text{lp}}) \triangleq \max_{\tau_h \in \mathbb{B}(i,A)}\{NPS_h - \varepsilon\} + e^{\text{lp}}$.

Considering e^{hep}, we can again account for its effects by introducing a hypothetical exceedance task τ_E that has higher priority than any other task in the workload. Reasoning analogously to the cases of EDF and FIFO, we conclude that the contribution of equal-or-higher-priority tasks to the maximum busy-window length L is bounded by $e^{\text{hep}} + \sum_{h \in \mathbb{HEP}^+(i)} RBF_h(L)$. Putting everything together, since $B_i'(A, e^{\text{lp}}) = B_i(A) + e^{\text{lp}}$, we arrive at the following overall exceedance-aware bound on the maximum busy-window length under FP scheduling:

$$L_i(e) \triangleq \min\{L > 0 \mid L \geq e + B_i(A) + \textstyle\sum_{\tau_h \in \mathbb{HEP}^+(i)} RBF_h(L)\}.$$

2) Exceedance-aware search space: The rationale underlying the sparse search space [16], as sketched in Sec. IV-B, is unaffected by NET exceedance. In fact, for reasons that will shortly become clear, the search space can still be defined in terms of IBF_i as before. However, we must account for potentially longer busy windows using the exceedance-aware bound $L_i(e)$, which leads to the following definition closely resembling the base case without exceedance: $\mathcal{A}_i(e) \triangleq \{0\} \cup \{0 < A < L_i(e) \mid \exists \Delta,\ IBF_i(A - \varepsilon, \Delta) \neq IBF_i(A, \Delta)\}$.

3) Exceedance-aware offset analysis: Let us now consider the problem of finding an exceedance-aware bound $F_i^A(e)$ on the length of J_i's preemptive part for a fixed arrival offset A.

First, any exceedance in the last non-preemptive segment of J_i (*i.e.*, the part of the job that runs to completion once scheduled) has only a linear effect on its response time since, by definition, J_i completes at the end of this segment. In contrast, any exceedance in earlier segments of J_i, or in any preceding jobs of the same or other tasks, can result in nonlinear effects.

TABLE V: Interference bound functions and blocking bounds for FP [16], EDF [4, 16], and FIFO [13].

Policy	$IBF_i(A, \Delta)$	$B_i(A)$
FP	$RBF_i(A+\varepsilon) - C_i + B_i(A) + \sum_{\tau_h \in \mathbb{HEP}(i)} RBF_h(\Delta)$	$\max_{\tau_h \in \mathbb{B}(i,A)}\{NPS_h - \varepsilon\}$
EDF	$RBF_i(A+\varepsilon) - C_i + B_i(A) + \sum_{\tau_h \neq \tau_i} RBF_h(\min\{A + \varepsilon + D_i - D_h, \Delta\})$	$\max_{\tau_h \in \mathbb{B}(i,A)}\{NPS_h - \varepsilon\}$
FIFO	$\left(\sum_{\tau_h \in \tau} RBF_h(A+\varepsilon)\right) - C_i$	$\emptyset$

Therefore, we assume, without loss of generality, that any exceedance manifests before the job under analysis reaches its run-to-completion threshold (*i.e.*, in the worst case, all exceedance occurs before J_i's final part begins).

With this premise, we can follow the same approach we used to bound $L_i(e)$. Our goal is to derive an extended $IBF_i{}'(A, \Delta, e)$ that accounts for exceedance. Let us consider FIFO scheduling first, as its definition of IBF_i is the simplest (recall Table V). Again attributing any exceedance to jobs of a fictitious task τ_E and exploiting $RBF_E(\Delta) = e$, we obtain $IBF_i{}'(A, \Delta, e) = \left(RBF_E(\Delta) + \sum_{\tau_h \in \tau} RBF_h(A+\varepsilon)\right) - C_i = \left(e + \sum_{\tau_h \in \tau} RBF_h(A+\varepsilon)\right) - C_i = IBF_i(A, \Delta) + e$.

Next, in the case of FP scheduling, we again split the total exceedance in two parts $e^{\text{hep}} + e^{\text{lp}} = e$. As before when bounding $L_i(e)$, the worst-case contribution by lower-priority jobs is accounted for by the exceedance-aware blocking bound $B_i'(A, e^{\text{lp}}) = B_i(A) + e^{\text{lp}}$. Using the fictitious task τ_E to model any higher-or-equal-priority exceedance e^{hep}, we obtain $RBF_i(A+\varepsilon) - C_i + e^{\text{hep}} + \sum_{\tau_h \in \mathbb{HEP}(i)} RBF_h(\Delta)$ as an upper bound on the contribution of equal-or-higher-priority tasks to total interference (including any exceedance in the job under analysis). Combining the two bounds, we arrive at $IBF_i{}'(A, \Delta, e) = RBF_i(A+\varepsilon) - C_i + B_i'(A, e^{\text{lp}}) + e^{\text{hep}} + \sum_{\tau_h \in \mathbb{HEP}(i)} RBF_h(\Delta) = RBF_i(A+\varepsilon) - C_i + B_i(A) + e + \sum_{\tau_h \in \mathbb{HEP}(i)} RBF_h(\Delta) = IBF_i(A, \Delta) + e$.

For the sake of brevity, we omit the derivation of $IBF_i{}'$ under EDF scheduling, which proceeds largely identically to the above FP case. Overall, we observe that $IBF_i{}'(A, \Delta, e) = IBF_i(A, \Delta) + e$ under all considered scheduling policies.

From this, in analogy to Eq. (1), *i.e.*, the base case without exceedance, the bound $F_i^A(e)$ on the length of the preemptive part of J_i can be obtained by solving the fixed point $A + F_i^A(e) = RCT_i + IBF_i{}'(A, A + F_i^A(e), e)$, which reduces to

$$\min_{F_i^A(e) > 0} \{A + F_i^A(e) = RCT_i + IBF_i(A, A + F_i^A(e)) + e\}. \quad (2)$$

4) Exceedance-aware response-time bound: Finally, wrapping up as in the exceedance-free base version, we define

$$F_i(e) \triangleq \max\{F_i^A(e) \mid A \in \mathcal{A}_i(e)\},$$

which yields the final bound $R_i(e) = F_i(e) + (C_i - RCT_i)$.

If $L_i(e)$ or $F_i^A(e)$ for any $A \in \mathcal{A}_i(e)$ cannot be found, then the initial assumption of the system not being fully utilized at nominal levels is violated and $R_i(e)$ is undefined.

V. Finding Nonlinearities

We now return to the problem of finding values of e corresponding to nonlinearities of $R_i(e)$. It is inherent in the nature of measured NETs that we cannot bound the maximum possible e, as that would imply a WCET analysis. Instead, engineers will examine EEs up to an application- and context-specific *exceedance safety margin*, after which EEs of even larger magnitude are deemed to be so unlikely as to be of negligible significance (*e.g.*, subsumed by the domain-specific acceptable level of residual risk). Setting the appropriate exceedance threshold is ultimately a judgement call driven by business as well as safety objectives, informed by engineering experience and domain expectations.

We thus seek to *iterate* over all such values of e that correspond to nonlinearities in $R_i(e)$, in increasing order, for as long as engineers are interested in exploring EEs of increasing magnitude. In particular, we do *not* assume that we are given a maximum threshold for e up front. Instead, we support a workflow in which engineers can ask for the *next-largest nonlinearity* as often as necessary for their goals.

More precisely, the goal is to enumerate (an arbitrary prefix of) a sequence of values e_y that satisfy the following property:

$$e_y \triangleq \begin{cases} 0 & \text{if } y = 0 \\ \min\{e \mid R_i(e) - R_i(e_{y-1}) > (e - e_{y-1})\} & \text{if } y \geq 1 \end{cases}$$

Search algorithm. As we demonstrate in Sec. VII, computing e_y with a brute-force search (*i.e.*, simply computing $R_i(e)$ for every $e \in \{1, 2, 3, \ldots\}$) is much too slow to be practical, due to two reasons: first, the numerical magnitudes of typical task parameters, which are commonly expressed at the granularity of processor cycles or nanoseconds, are too large; and second, nonlinearities are usually sparse in the exceedance domain (*e.g.*, even using a coarse time unit, more than 90% of the exceedance values shown in Fig. 2 cause only a linear increase).

We instead use an *exponential search* to quickly find intervals containing at least one nonlinearity, and then a *binary search* to identify individual nonlinearities. The resulting procedure iterating over all nonlinearities is given in Algorithm 1.

The procedure has two parts: the first part (lines 6–17) realizes the initial exponential search for an interval containing one or more nonlinearities, and the second part (lines 19–31) implements the binary search that identifies individual nonlinearities. The algorithm uses two state variables, *done* and *unex*. The former, *done*, strictly lower-bounds where the next element of e_y may be found and holds the corresponding response-time bound. Its purpose is to keep track of the progress made by the search so far and is hence initialized to the exceedance-free base case (line 2). The latter state variable, *unex*, is a stack of unexplored intervals containing at least one nonlinearity. Initially, it is empty (line 3).

The search is parametrized by two heuristics that steer the exponential search: *step*, which is the base step size taken by the exponential search (in line 9), and *retry_limit*, which is

Algorithm 1 RTA Nonlinearity Search

```
1:  procedure NONLINEARITY ITERATOR($step$, $retry\_limit$)
2:      $done \leftarrow (0, R_i(0))$                        ▷ init. progress marker
3:      $unex \leftarrow \emptyset$                             ▷ init. exploration stack
4:      while true do
5:              ▷ Part 1: exponential nonlinearity interval search
6:          $attempt \leftarrow 0$
7:          while $unex = \emptyset$ and $attempt \leq retry\_limit$ do
8:              $e^l, R^l \leftarrow done$
9:              $e^r \leftarrow e^l + step \cdot 2^{attempt}$              ▷ exponential step
10:             $R^r \leftarrow R_i(e^r)$                          ▷ compute RTA
11:             if $R^r - R^l > e^r - e^l$ then                ▷ nonlinear step?
12:                 push $((e^l, R^l), (e^r, R^r))$ onto $unex$
13:             else
14:                 $done \leftarrow (e^r, R^r)$             ▷ jump over linear space
15:                 $attempt \leftarrow attempt + 1$
16:         if $unex = \emptyset$ then
17:             return "none found"                    ▷ terminate search
18:             ▷ Part 2: binary search for nonlinearity in interval
19:         while true do
20:             $((e^l, R^l), (e^r, R^r)) \leftarrow$ pop from $unex$
21:             if $e^l + \varepsilon = e^r$ then              ▷ is it a singleton range?
22:                 yield nonlinearity $e^r$                     ▷ found one
23:                 $done \leftarrow (e^r, R^r)$                    ▷ mark progress
24:                 break                                ▷ continue in Line 4
25:             else                                   ▷ split interval in two
26:                 $e^m \leftarrow e^l + \left\lfloor \frac{e^r - e^l}{2} \right\rfloor$                 ▷ find midpoint
27:                 $R^m \leftarrow R_i(e^m)$                      ▷ compute RTA
28:                 if $R^r - R^m > e^r - e^m$ then           ▷ nonlinear?
29:                     push $((e^m, R^m), (e^r, R^r))$ onto $unex$
30:                 if $R^m - R^l > e^m - e^l$ then           ▷ nonlinear?
31:                     push $((e^l, R^l), (e^m, R^m))$ onto $unex$
```

the maximum exponent used before giving up (lines 7 and 17). We propose generally suitable defaults for these further below.

The exponential search is triggered whenever $unex$ is empty (line 7). It repeatedly checks an interval of length $step \cdot 2^{attempt}$ starting at $done$, where $attempt$ is the number of unsuccessful search steps in the current iteration. It terminates either when a nonlinearity-containing interval is found (and pushed onto $unex$, line 12) or when $attempt$ reaches $retry\_limit$. Unsuccessful attempts advance $done$ (line 14). If all $retry\_limit$ attempts fail, the entire iteration procedure terminates (lines 16–17), based on the idea that, for reasonable choices of $step$ and $retry\_limit$, there likely are no more nonlinearities to be found.

Once $unex$ is nonempty, the binary search proceeds by repeatedly splitting the top element of $unex$ (lines 26–27) and pushing one or both halves containing nonlinearities back onto $unex$ (lines 28–31). The splitting continues until the top element is a singleton interval containing only one point (line 21), which is then exactly the next element of e_y (line 22).

Default heuristics. For Algorithm 1 to work well, it is essential to pick default values for $step$ and $retry\_limit$ suitable for a wide variety of workloads. We have found that the following configuration works well (for the setup considered in Sec. VII).

For a periodic workload τ, we use $step = \max\{T_j | \tau_j \in \mathbb{HEP}^+(i)\} \cdot (1 - \sum_{\tau_j \in \mathbb{HEP}^+(i)} C_j / T_j)$, rounded to the nearest integer. This parameter choice is important for scaling the step size with the amount of available static slack. If the workload has little slack at nominal utilization levels, it has only limited ability to dissipate transient overload, so the exponential search should be careful to initially take only relatively small steps. Otherwise, the busy-window bound $L_i(e)$ (and hence the search space $\mathcal{A}_i(e)$) can quickly become quite large, thereby slowing down the search. Multiplying by the maximum period of any interfering task ensures that the heuristic works well irrespective of the input's time-unit granularity. Finally, for $step$ as defined above, we set $retry\_limit = 14$, which in our testing did not exhibit any false positives (*i.e.*, early termination).

VI. Exceedance Allocation

For an engineer who is assessing a system's robustness to execution-time uncertainty, the basic information that a certain amount of total exceedance e *may* cause a nonlinear increase in the response time of a given task τ_i is rather abstract and difficult to interpret by itself. Instead, it is much more helpful and intuitive to examine example traces that explain *how* the nonlinear increase could arise. However, generating useful traces is challenging, for two reasons: First, to obtain a *valid* trace, the total exceedance e must be divided among individual jobs so the resulting busy window is long enough to contain the release offset A that yields the maximum response time $R_i(e)$. Second, to obtain *meaningful* examples, exceedance should be allocated to jobs in a plausible manner consistent with the engineer's system-specific expertise and knowledge.

To address both challenges, we formulate exceedance allocation as an optimization problem. In the following, we first present basic constraints that ensure trace validity, and then add "tuning knobs" that allow engineers to steer the generated traces towards more realistic and relevant scenarios.

A. Setup and Basic Validity Constraints

Given a fixed amount of total exceedance e, the task under analysis τ_i, and the release offset $A \in \mathcal{A}_i(e)$ that yields the maximum response time $R_i(e)$, we use *mixed-integer linear programming* (MILP) to allocate exceedance to all jobs that feature in a busy window of length $L_i(e)$. In the following, we assume that in the generated trace jobs are released as rapidly as allowed by each task's arrival curve and that every job consumes its task's full NET (plus any allocated exceedance).

Under these assumptions, we let $\mathrm{Q}_h^{\mathrm{HP}}$ denote the number of higher-or-equal-priority jobs of task τ_h that can delay τ_i's job released at time A (recall Eq. (2) and $IBF_i(A, \Delta)$ in Table V):

$$\mathrm{Q}_h^{\mathrm{HP}} \triangleq \begin{cases} q_h^{\mathrm{HP}} & \text{if } \tau_h \in \mathbb{HEP}(i) \\ \alpha_h(A + \varepsilon) & \text{if } \tau_h = \tau_i \\ 0 & \text{otherwise,} \end{cases}$$

where, for FP, $q_h^{\mathrm{HP}} = \alpha_h(A + F_i^A(e))$, for FIFO, $q_h^{\mathrm{HP}} = \alpha_h(A + \varepsilon)$, and for EDF, $q_h^{\mathrm{HP}} = \alpha_h(\min\{A + \varepsilon + D_i - D_h, A + F_i^A(e)\})$.

Based on $\mathrm{Q}_h^{\mathrm{HP}}$, we introduce the main optimization variables: $\forall \tau_h \in \tau$, $\forall\ j \in \{1, \ldots, \mathrm{Q}_h^{\mathrm{HP}}\}$, the *higher-or-equal-priority exceedance* $\mathrm{x}_{h,j}^{\mathrm{HP}}$ is the amount of exceedance exhibited by an interfering job $J_{h,j}$; and $\forall \tau_h \in \mathbb{B}(i, A)$, the *lower-priority exceedance* $\mathrm{x}_h^{\mathrm{B}}$ is the amount of exceedance allocated to a job of τ_h causing priority inversion (there is at most one such job). The total exceedance must be distributed across these variables.

Constraint 1. $\sum_{\tau_h \in \tau} \left(\mathrm{x}_h^B + \sum_{j=1}^{\mathrm{Q}_h^{HP}} \mathrm{x}_{h,j}^{HP} \right) = e$

To control lower-priority exceedance, we introduce additional binary indicator variables $\forall \tau_h \in \mathbb{B}(i, A)$, $\mathrm{s}_h^{\mathrm{B}} \in \{0, 1\}$ such that, if $\mathrm{s}_h^{\mathrm{B}} = 0$, then $\mathrm{x}_h^{\mathrm{B}} = 0$, and if $\mathrm{x}_{h,j}^{\mathrm{HP}} > 0$ for any j, then $\mathrm{s}_h^{\mathrm{B}} = 0$ (which both can easily be stated in linear form using standard techniques). There is at most one blocking task.

Constraint 2. $\sum_{\tau_h \in \mathbb{B}(i,A)} \mathrm{s}_h^B \leq 1$

In the case of workloads with floating non-preemptive sections, not all tasks may have non-preemptive sections. Thus, if some task τ_h does not have any non-preemptive sections, then $\mathrm{s}_h^{\mathrm{B}}$ is necessarily set to 0. Similarly, in the case of fully preemptive tasks, $\mathrm{s}_h^{\mathrm{B}} = 0$ for all tasks (equivalently, blocking-related variables can simply be omitted for this preemption model).

We next address the problem that the busy window must not end prematurely and that the final non-preemptive segment of τ_i must not start executing until *after* all potentially interfering higher-priority jobs have been released. In other words, exceedance allocated to jobs that are released after τ_i is guaranteed to run to completion would have no effect.

To express this constraint concisely, we use the well-known *minimum-distance function* $\delta_h(n) \triangleq \min_\Delta\{\alpha_h(\Delta) \geq n\}$ [40], a pseudo-inverse of the arrival curve α_h that yields the shortest interval in which n consecutive jobs of τ_h arrive. For $j \geq 1$, we let $a_{h,j}^* = \delta_h(j) - \varepsilon$ denote $J_{h,j}$'s release offset in the busy window. We further define $\mathbb{J}_h(j) \triangleq \{J_{m,n} | \, a_{m,n}^* < a_{h,j}^*, n \leq Q_m^{\mathrm{HP}}\}$ to represent all jobs that are released strictly before $a_{h,j}^*$.

Consider any interfering job $J_{h,j}$: we seek to express that both the busy window has not ended and the final non-preemptive segment of τ_i has not yet started when $J_{h,j}$ is released at time $a_{h,j}^*$. This requires that the total processor demand since the beginning of the busy window (including any exceedance) has not yet been met, *i.e.*, the total demand of all higher-or-equal-priority jobs released in $[0, a_{h,j}^*)$ and any lower-priority blocking must exceed the interval's length $a_{h,j}^*$.

The total demand (including any exceedance) of higher-or-equal-priority jobs released prior to $a_{h,j}^*$ is given by $\sum_{\tau_k \in \mathbb{HEP}^+(i)} \sum_{J_{k,l} \in \mathbb{J}_h(j)} C_k + \mathrm{x}_{k,l}^{\mathrm{HP}}$. Additionally, the delay due to lower-priority exceedance at the start of the busy window is given by $\sum_{\tau_k \in \mathbb{B}(i,A)} \mathrm{x}_k^{\mathrm{B}}$. This is on top of the nominal blocking bound $B_i(A)$, which also must be accounted for. Finally, if $a_{h,j}^* \geq A$, the final non-preemptive segment of τ_i is implicitly included in the demand of tasks in $\mathbb{HEP}^+(i)$, which we correct by subtracting $C_i - RCT_i$ (*i.e.*, the length of τ_i's last segment). Combining everything, we arrive at the following constraint.

Constraint 3. $\forall \tau_h \in \mathbb{HEP}^+(i), \forall j \in \{2, \ldots, \mathrm{Q}_h^{HP}\}$,

$$\left(\sum_{\tau_k} \in \mathbb{HEP}^+(i) \sum_{J_{k,l} \in \mathbb{J}_h(j)} C_k + \mathrm{x}_{k,l}^{HP}\right) + \left(B_i(A) + \sum_{\tau_k \in \mathbb{B}(i,A)} \mathrm{x}_k^B\right) - LS \geq a_{h,j}^* + \varepsilon,$$

where $LS = (C_i - RCT_i)$ *if* $a_{h,j}^* \geq A$ *and* $LS = 0$ *otherwise.*

The constraint does not apply for $j = 1$ because the first job of each task in $\mathbb{HEP}^+(i)$ is always part of the busy window.

There is an additional subtlety related to the task selected to cause priority inversion (*i.e.*, the task τ_h for which $\mathrm{s}_h^{\mathrm{B}} = 1$, if any). If this task's longest non-preemptive section is not maximal (*i.e.*, $NPS_h < B_i(A)$), then *additional* exceedance (not accounted for by e) is required to "fill" the task's *blocking margin*, which is defined as $bm_h = B_i(A) - NPS_h$. That is, the generated scenario actually requires not just e, but a total of $e + \sum_{\tau_h \in \mathbb{B}(i,A)} \mathrm{s}_h^{\mathrm{B}} \cdot bm_h$ exceedance to take place (*i.e.*, picking a non-maximal non-preemptive section *shifts* the scenario in the exceedance space by bm_h time units). In situations where this is undesirable, it can be avoided simply by forcing $\mathrm{s}_h^{\mathrm{B}} = 0$ for each $\tau_h \in \mathbb{B}(i, A)$ with $bm_h > 0$, which prevents non-maximal non-preemptive sections from being chosen.

As we have not yet specified an objective function; the above constraints can be reused for any desired optimization criteria. In fact, the constraints can be trivially satisfied by allocating all of e to (one of) the higher-or-equal-priority job(s) released at the beginning of the busy window (time 0). However, depending on a system's characteristics, such a scenario—a single job being responsible for *all* exceedance—may not be particularly plausible or interesting, especially for large values of e. We thus next introduce a configurable optimization objective to steer the solver towards more relevant solutions.

B. Tailoring Traces to System-Specific Expertise

Trace generation is influenced by the following inputs, which are intended to be "tunable knobs" for engineers to express assumptions about their specific system under evaluation.

- *NET trustworthiness* Θ_h: for each task τ_h, $\Theta_h \in [0, 1]$ indicates the engineer's subjective level of trust in the correctness of τ_h's NET (0 meaning "not trusted at all" and 1 meaning "highly trusted"). For example, the NET of a recently developed task may be seen with suspicion.
- *Exceedance balancer* V: the value $\mathrm{V} \in [0, 1]$ indicates how much the optimizer tries to balance the exceedance among all tasks rather than attributing it to just a few of them (the higher V, the lower the number of tasks involved). For example, memory interference is likely to slow down multiple tasks at the same time, but other sources of execution-time uncertainty can have more localized effects.
- *Maximum and minimum exceedance* MAX_h and min_h: For each task τ_h, $\mathrm{MAX}_h \geq 0$ and $\mathrm{min}_h \geq 0$ limit the maximum and minimum exceedance that can be allocated to each job of τ_h to $\mathrm{MAX}_h \cdot C_h$ and $\mathrm{min}_h \cdot C_h$, respectively. For example, MAX_h can be set to zero if a task τ_h has an enforcement mechanism that aborts overrunning jobs.

Some combinations of these parameters may produce optimization problems with empty feasible regions. In this case, the user should be warned and asked to change the configuration.

The impact of maximum and minimum exceedance is clear.

Constraint 4. $\forall \tau_h \in \tau$, $min_h \cdot C_h \leq \mathrm{x}_h^B \leq MAX_h \cdot C_h$; *and* $\forall j \in \{1, \ldots, \mathrm{Q}_h^{HP}\}$, $min_h \cdot C_h \leq \mathrm{x}_{h,j}^{HP} \leq MAX_h \cdot C_h$.

The other parameters appear only in the objective function, which requires additional elaboration. The overall goal is to:

$$\textbf{minimize} \quad \beta_\Theta \cdot f^{\mathrm{TH}} + \beta_{\mathrm{V}} \cdot f^{\mathrm{EB}} + \beta_{\mathrm{s}} \cdot f^{\mathrm{BL}}.$$

The first two weighted parts express penalties related to the tuning parameters: **(1)** a *trustworthiness cost* f^{TH} and **(2)** an *exceedance-balancer cost* f^{EB}. Additionally, **(3)** a *blocking cost* f^{BL} guides which task causes priority inversion (if any). The weights β_Θ, β_V, and β_s can be chosen arbitrarily (*e.g.*, $\beta_\Theta + \beta_\text{V} + \beta_\text{s} = 1$). We next explain each part in turn.

First, f^{TH} considers the exceedance allocated to interfering jobs, weighted by the NET trustworthiness parameters Θ_j.

$$f^{\text{TH}} = \sum_{\tau_h \in \tau} \Theta_h \left[\sum_{j=1}^{\text{Q}_h^{\text{HP}}} \left(1 + \frac{\text{x}_{h,j}^{\text{HP}}}{\sqrt{C_h}}\right)^2 + \left(1 + \frac{\text{x}_h^{\text{B}}}{\sqrt{NPS_h}}\right)^2 \right]$$

The allocated exceedance is normalized w.r.t. each task's nominal bounds. The components of the sum are squared to encourage exceedance to be distributed across multiple jobs when possible (rather than concentrating exceedance in just one job per task), and a constant 1 is added to ensure that the cost grows monotonically with increasing exceedance. While squaring the per-job variables $\text{x}_{h,j}^{\text{HP}}$ and x_h^{B} technically moves the optimization problem beyond the reach of MILP solvers, in our experience it is sufficient and effective to use a piecewise-linear approximation of the objective function.

Next, $f^{\text{EB}} = \text{V} \cdot \sum_{\tau_h \in \tau} \text{s}_h^{\text{B}} + \sum_{j=1}^{\text{Q}_h^{\text{HP}}} \text{s}_{h,j}^{\text{HP}}$ simply adds a penalty proportional to the number of jobs exceeding nominal bounds, weighted by the balancer setting V. Here, $\text{s}_{h,j}^{\text{HP}}$ is a new binary indicator variable such that $\text{s}_{h,j}^{\text{HP}} = 1$ iff $\text{x}_{h,j}^{\text{HP}} > 0$.

Finally, $f^{\text{BL}} = \sum_{\tau_h \in \mathbb{B}(i,A)} \text{s}_h^{\text{B}} \cdot bm_h$ adds a penalty if the task τ_h selected to cause blocking is not (one of) the task(s) with the maximal non-preemptive section length accounted for by $B_i(A)$. The rationale for this penalty is that such a task τ_h requires extra exceedance to cover its blocking margin bm_h, which as already discussed complicates the picture.

As shown in Sec. VII, the proposed "tuning knobs" are effective in generating a wide variety of sample traces. Nonetheless, it is worth noting that the objective function proposed in this section is only one of many ways in which the optimizer can be steered to produce traces with desired characteristics. As such, it is easily possible to tweak or completely replace the optimization criteria as desired in a given application context.

VII. Evaluation

We evaluated Algorithm 1 in terms of its efficiency and the trace generator's ability to respect tuning parameters.

Workloads. We generated synthetic task sets using two methods: **(i)** the *Dirichlet-Rescale* (DRS) algorithm [33] and **(ii)** according to an automotive benchmark [44], which hereafter we refer to as *DRS* and *W15* workloads, respectively.

DRS workloads were generated by varying the total utilization $U \in \{0.2, 0.3, \ldots, 0.9, 0.95, 0.99\}$ and the number of tasks $n \in \{5, 10, \ldots, 50\}$. For each combination of U and n, 10 task sets were generated, for a total of 1000 task sets for each of the four considered preemption models (recall Table II). Given U and n, the DRS algorithm [33] was used to generate a vector $u_1, \ldots, u_n$ of nominal per-task utilizations summing to U. The period T_i for each task τ_i was drawn log-uniformly from 1 ms to 1000 ms and the NET derived as $C_i = u_i \cdot T_i$. For limited-preemptive workloads, the number of segments was drawn uniformly at random from $\{3, \ldots, 15\}$. The DRS algorithm was then used again to derive the NET of each segment. For floating non-preemptive workloads, we drew a fraction y_i uniformly at random from $[0.05, 0.15]$ and set $\gamma_i = y_i \cdot \min\{C_i, \text{median}\{C_j\}_j\}$ to avoid excessive blocking.

W15 workloads were obtained by generating 100 task sets for each $n \in \{5, 6, \ldots, 40\}$, for a total of 3600 task sets, drawing per-task utilizations and periods as specified by the benchmark [44]. W15 workloads are fully preemptive.

Following the automotive challenge by Hamann et al. [35], the basic time unit ε was set to 1 clock cycle assuming a 200 MHz processor (for both DRS and W15 workloads). Finally, we used FP scheduling and assigned rate-monotonic priorities.

Nonlinearity search. We compared Algorithm 1 against a *brute-force baseline* that identifies response-time nonlinearities by computing $R_i(e)$ for $e \in \{1, 2, 3, \ldots\}$. The experiments were run on an Intel Xeon Platinum 8180 CPU @ 2.50 GHz with 56 physical cores and 112 hardware threads. The implementation is a sequential Python script; we ran one instance per hardware thread to process multiple workloads in parallel. For each generated workload, Algorithm 1 and the baseline were each run for five minutes, counting the number of nonlinearities found for all tasks. Figs. 3 and 4 show representative results.

Fig. 3(a) compares the maximum, mean, and minimum number of nonlinearities found by the two approaches for 100 DRS task sets, each composed of 25 fully non-preemptive tasks (and with varying total utilization). While the brute-force baseline could find only at most 1 nonlinearity per task in 5 minutes, Algorithm 1 found, on average, more than 700 nonlinearities (with a maximum of 1418 and a minimum of 372). Similar trends can be observed under other configurations in Fig. 3(b) and Fig. 3(c). Fig. 3(b) shows results for 100 DRS task sets composed of a variable number of fully preemptive tasks with total utilization $U = 0.7$. Finally, Fig. 3(c) shows that the same trends arise for W15 workloads, too.

Whereas Fig. 3 shows the number of nonlinearities found over time, Fig. 4 reports the total found after 5 minutes. Fig. 4(a) reports the results for the floating non-preemptive preemption model, grouped by n (with varying U). Algorithm 1 again clearly outperforms the brute-force baseline for every tested task-set size. Under both approaches, the number of nonlinearities found in 5 minutes decreases with increasing n. This is because Fig. 4 reports the number of nonlinearities found *per task*, so that more nonlinearities must be found in total for larger n. Additionally, computing $R_i(e)$ becomes slower with increasing n. Fig. 4(b) reports results for limited-preemptive tasks, aggregated by U (with varying n). For these workloads, the brute-force baseline never found more than 13 nonlinearities (for $U = 0.99$). Our analysis instead found, on average, more than 600 nonlinearities, with a maximum of 6900 for $U = 0.9$. Similar results were observed with W15 workloads.

Overall, our results show Algorithm 1 to be **(i)** *necessary*, as a brute-force approach is clearly too slow in practice, and **(ii)** *practical*, as it yields dozens to hundreds of nonlinearities for realistically sized workloads in mere seconds.

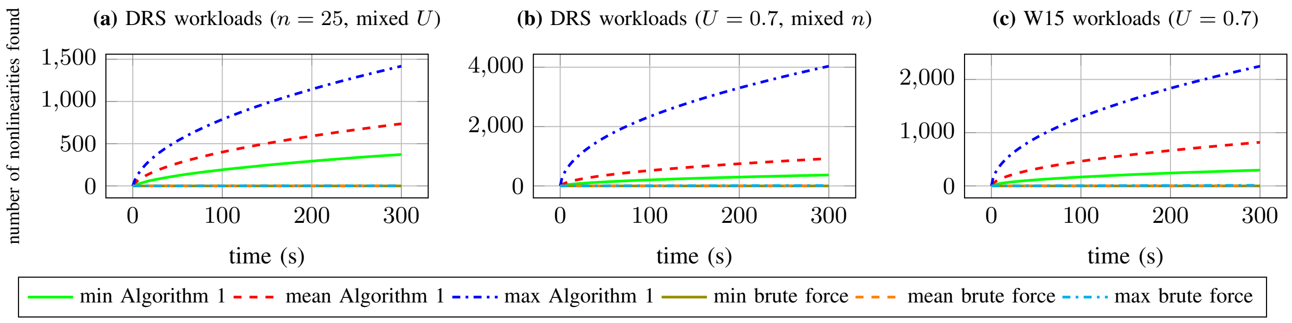

Fig. 3: Nonlinearities found *vs.* runtime for **(a)** fully non-preemptive DRS, **(b)** fully-preemptive DRS, and **(c)** W15 task sets.

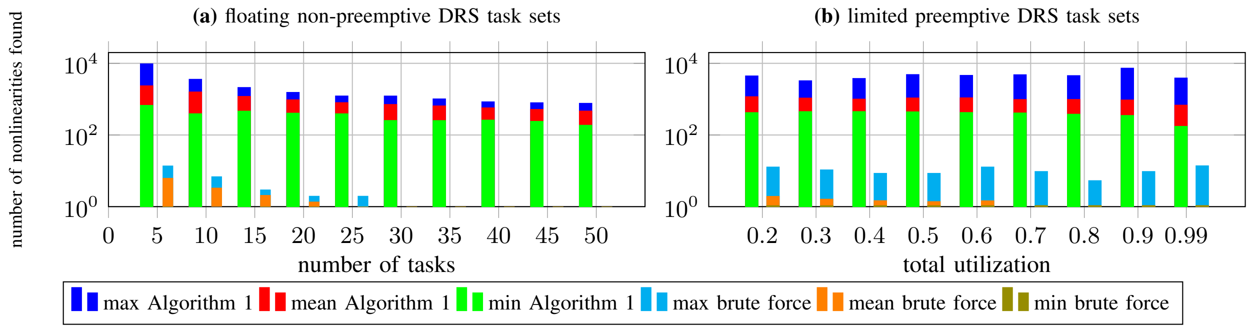

Fig. 4: Number of nonlinear steps found as a function of the number of tasks and total utilization of the task set.

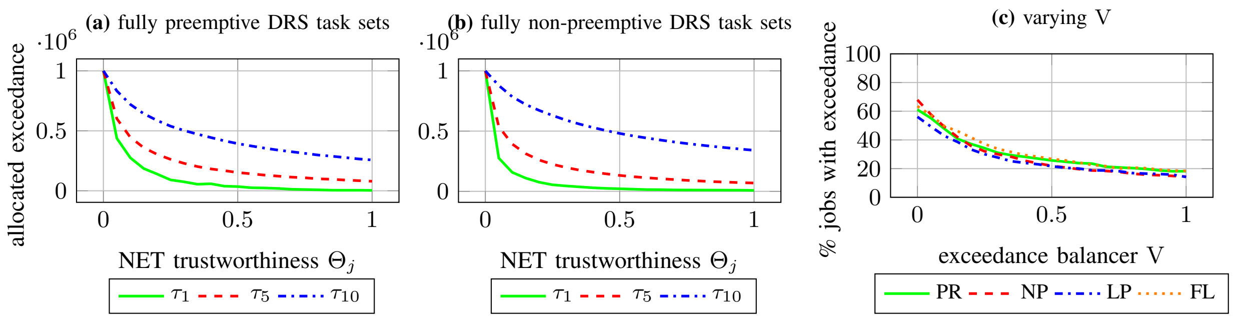

Fig. 5: Effects of tuning parameters. PR: preemptive; NP: non-preemptive; LP: limited preemptive; FL: floating non-preemptive.

Trace generation. We evaluated the trace generator (Sec. VI) to assess the impact of the proposed "tuning knobs" on the generated example traces. As a representative subset, we focused on DRS task sets composed of 10 tasks (for a total of 100 task sets, 10 for each considered value of U). The MILP was implemented with the Pyomo [21, 39] modeling framework for Python and solved with Gurobi Optimizer (version 10.0.3).

As already mentioned in Sec. VI-B, we let Pyomo convert the quadratic objective function into a piecewise-linear approximation by sampling the quadratic terms for 1000 evenly spaced values of $\mathrm{x}^{\mathrm{HP}}_{h,j}$ and $\mathrm{x}^{\mathrm{B}}_{h}$ across $[0, e]$, thus obtaining a MILP.

The weights β_{s}, β_{V}, and β_{Θ} of the objective function were selected as follows. We first simulated a trace in which each interfering job is allocated an amount of exceedance proportional to its NET. Then, using this distribution of exceedance as a reference, we computed the three weights such that $\beta_\Theta + \beta_{\mathrm{s}} + \beta_{\mathrm{V}} = 1$ and $\beta_\Theta \cdot f^{\mathrm{TH}} = \beta_{\mathrm{s}} \cdot f^{\mathrm{BL}} = \beta_{\mathrm{V}} \cdot f^{\mathrm{EB}}$. The rationale behind this choice is to study a configuration of the weights wherein all three components of the objective function make an equal contribution to the overall cost to minimize.

We evaluated the impact of the NET trustworthiness parameter Θ_i by varying it from 0 to 1 in steps of 0.05, for one task at a time, while allocating a fixed amount of exceedance $e = 10^6$ clock cycles. The optimization problem was solved for the job of the lowest-priority task with arrival offset $A = 0$.

Figs. 5(a) and (b) show results for respectively fully preemptive and fully non-preemptive workloads. Each graph shows three curves representing different tasks across the priority spectrum for which the NET trustworthiness parameter is varied (here, τ_1 has the highest priority). In all cases, the desired effect is observed: as a task's trustworthiness increases, the exceedance allocated to it diminishes. Notably, lower-priority tasks attract more exceedance since f^{TH} normalizes allocations w.r.t. NETs and the DRS workload generator tends to produce tasks the longer the period of a task, the higher its NET.

Next, we varied the balancer setting V from 0 to 1 in steps of 0.05 to test its effect on the number of jobs with nonzero exceedance. Fig. 5(c) reports the observed results for all four considered preemption models (averaged across all tested task sets). As expected, fewer jobs are allocated exceedance as V increases. Whereas $\approx 60\%$ of jobs exceed their NETs for $\mathrm{V} = 0$, less than 20% do so for $\mathrm{V} = 1$.

Overall, the results in Fig. 5 confirm that the proposed configuration parameters effectively control the trace characteristics.

VIII. Case Study

To explore the practical use of exceedance analysis, we extracted a task set from the well-known WATERS'17 industrial challenge [35], which models an automotive workload provisioned on four cores clocked at 200 MHz under partitioned FP scheduling. We focus here on the periodic, implicit-deadline workload on core 2 (see Table VI), which (in)famously is unschedulable with the maximum execution costs specified by the challenge, as the tasks' total utilization exceeds one [58, 64]. In other words, in the challenge's reference system, tasks are provisioned based on NETs, and not on the basis of WCETs.

In the following, we therefore assume NETs at the 90% level, which ensures schedulability. More precisely, by applying a regular, exceedance-oblivious analysis assuming NETs equal to 90% of the reported maxima [35], it can be shown that each task's resulting NRT is less than its period. However, such NRTs are clearly not hard guarantees because **(i)** the system is knowingly not provisioned on a worst-case basis and **(ii)** the NETs in Table VI do not reflect cross-core memory contention, which is specified separately in the WATERS'17 challenge.

It is thus fully expected *that* NET overruns will occasionally occur at runtime, but it is far from obvious *how* susceptible individual tasks are. Specifically, it is not clear how much of a slowdown is necessary to induce a deadline miss in any of the tasks, nor how quickly the system can recover afterwards.

Exceedance. To answer these questions, we applied Algorithm 1 to the task set in Table VI and determined for each task τ_i the least amount of exceedance e such that $R_i(e) > D_i$. While Table VI reports milliseconds for readability, the analysis was carried out at the full precision of processor cycles.

The results are given in column e of Table VI. For example, it takes at least ≈ 3.6 ms of total exceedance for τ_3 to miss a deadline, which is slightly less than the total exceedance that τ_5 can tolerate, but considerably more than τ_1's exceedance margin. Without the analysis presented in Sec. IV-C, it would be difficult to obtain these margins, which are an essential prerequisite for further analysis of the system's temporal robustness.

Slowdown. That said, the raw exceedance thresholds are difficult to interpret and not directly comparable. We thus applied the optimization framework from Sec. VI to translate the total exceedance e into a *necessary slowdown*. Specifically, we reused the setup and basic constraints from Sec. VI-A, and then defined for each task τ_h a new variable x_h^s and constraints such that $x_h^s \geq \mathrm{x}_h^{\mathrm{B}} / C_h$ and $x_h^s \geq \mathrm{x}_{h,j}^{\mathrm{HP}} / C_h$ for each $j \in \mathrm{Q}_h^{\mathrm{HP}}$. The variable x_h^s represents the *maximum slowdown* among τ_h's jobs. The objective was then set to *minimize* $\max_{\tau_h \in \tau} x_h^s$.

The obtained value, reported as $\min \max x_h^s$ in Table VI, is the necessary slowdown required for a deadline miss to occur.

TABLE VI: Case Study Workload and Analysis Results

Task	Period	NET	e	$\min \max x_h^s$	$L_7(e)$
τ_1	2 ms	0.364 ms	1.636 ms	450.06%	97.59 ms
τ_2	5 ms	0.838 ms	3.071 ms	159.24%	99.39 ms
τ_3	20 ms	9.421 ms	3.591 ms	21.89%	99.91 ms
τ_4	50 ms	2.776 ms	14.407 ms	16.99%	399.06 ms
τ_5	100 ms	8.476 ms	3.929 ms	4.09%	179.91 ms
τ_6	200 ms	0.124 ms	7.733 ms	4.02%	279.91 ms
τ_7	1000 ms	0.123 ms	38.542 ms	4.01%	1079.91 ms

It paints a much clearer picture of the workload's resilience. For example, task τ_3 misses a deadline only if some jobs in the EE leading up to the deadline miss experience a slowdown of *at least* 21.89% due to memory contention or any other unusual runtime conditions. If there is less actual slowdown at runtime, then τ_3 is safe, *i.e.*, the derived *necessary* slowdown threshold is a direct measure of a task's temporal safety margin.

The difference in resilience among the tasks is immediately apparent. For example, the highest-priority task τ_1, by virtue of not suffering interference from any other task, would have to experience a massive 450% slowdown before missing a deadline. Conversely, tasks τ_5, τ_6, and τ_7 are revealed to be much more at risk. Since higher-priority interference leaves them little slack, the necessary slowdown is a mere $\approx 4\%$ for each of them. For instance, τ_5 can miss a deadline if the tasks $\tau_1 \ldots, \tau_5$ experience a maximum slowdown of only 4.09% across an EE spanning 100 ms (*i.e.*, τ_5's period), which is clearly not a big safety margin. Equipped with this information, engineers can decide whether so little leeway is acceptable (*e.g.*, if the tasks are not critical or time-sensitive) or if a redesign is in order to give the lower-priority tasks more slack.

For example, one potential approach could be to *buffer* activations, *i.e.*, allowing tasks to become temporarily backlogged without "losing" input. For some tasks, this can be an effective mitigation, while for others it is not: exceedance analysis reveals that, if τ_3 can buffer 3 activations (*i.e.*, $D_3 = 4 \cdot T_3$), then its minimum slowdown threshold rises to a comfortable 114.07%, but even assuming an extreme buffer of 6 activations, the slowdown threshold of τ_5 remains a meager 12.61%.

Recovery time. When deliberating the impact of lost activations, it can be helpful to consult the last column of Table VI, which lists $L_7(e)$ for the value of e given in the same row. Notably, $L_7(e)$—a bound on the maximum busy-window length of the lowest-priority task τ_7—also limits the maximum length of an EE with the stated total exceedance e.

For example, in a busy window in which τ_5 just barely misses a deadline (≥ 100 ms after the start of the busy window), it takes afterwards at most ≈ 80 ms for the system to dissipate the overload and reach a quiet time (*i.e.*, for the EE to end), assuming no further exceedance arises.

Conversely, with only minor changes to the optimization problem, one can infer the minimum slowdown necessary for transient overload to last longer than a given duration. For example, it takes a slowdown of at least 14.16% for task τ_4 to suffer an EE of length $L_3(e) > 1000$ ms. As 14.16% is less than τ_4's necessary slowdown for a deadline miss (16.99%), we infer that τ_4 would not yet miss a deadline despite the long EE.

IX. Discussion and Related Work

The proposed exceedance analysis is a means of studying a system's temporal behavior under any amount of transient overload (Secs. IV-C, V and VI), and can be used to infer its safety margins, *i.e.*, necessary conditions for undesired behavior (Sec. VIII). It does *not*, however, provide a bound on the maximum amount of total exceedance possible in any EE, nor does it rule out permanent overload. Whether a system's temporal safety margins are sufficient is an engineering decision that system designers must make on a case-by-case basis, and permanent overload must be ruled out via other means. For example, in the case of the workload studied in Sec. VIII, the reported *average* execution times [35] are well below the NETs given in Table VI, which precludes permanent overload.

The classic approach to dealing with WCET uncertainty is *sensitivity analysis*, first developed by Lehoczky et al. [49] for implicit-deadline tasks under rate-monotonic FP scheduling and later extended to constrained deadlines and any FP policy by Vestal [69], Punnekkat et al. [59], Yerraballi et al. [73], and Bougueroua et al. [15]. Bini et al. [14] developed the mathematically most elegant and efficient solution, which simultaneously allows for uncertainty in periods. Their work was later extended to the mixed-criticality setting by Dorin et al. [27]. More recently, George and Hermant [31] and Zhang et al. [74] provided sensitivity analysis for EDF. Chen et al. [23] focused instead on strictly periodic tasks. In contrast to this paper, none of the just-cited analyses supports limited-preemptive or fully non-preemptive tasks, none allows for arbitrary arrival curves, and none supports more than one scheduling policy. Furthermore, none of the sensitivity analyses for FP scheduling [14, 15, 27, 49, 59, 69, 73] supports arbitrary deadlines, all of which underlines our method's generality.

More significantly, a fundamental difference to exceedance analysis is that sensitivity analysis yields a scaling factor (or, depending on the specific method, an additive constant) by which one or more WCET parameters can be increased *indefinitely* while preserving schedulability. That is, sensitivity analysis uses *permanent* overload as the limiting criterion when maximizing tolerable WCET magnitudes. In contrast to exceedance analysis, it thus cannot yield insights into system behavior during *transient* episodes in which NETs are *temporarily* exceeded by arbitrary amounts above longterm sustainable levels. Sensitivity analysis also does not reveal response-time nonlinearities before or after task deadlines.

Several authors [11, 28, 36, 37, 60, 61, 72] studied workload models in which tasks are characterized by both an actual WCET (applicable to all jobs) and a second, lower bound that applies to a majority (but not all) of a task's jobs, which allows modeling (bounded) overload behavior. Stigge et al.'s expressive *digraph task model* [66] can similarly be used to represent tasks with "regular" and "unusual" computation needs. Ultimately, such models still rely on trusted WCETs to characterize the "unusual" demand, which we avoid here.

More closely related in spirit are techniques proposed by Racu et al. [62, 63], who explore the effect of arbitrary system model parameters on schedulability using binary search and evolutionary algorithms, and by Kumar and Thiele [45], who proposed a technique to find the *settling time* (longest interval in which the deadlines are missed) and *overshoot* (maximum number of jobs missing deadlines) after transient overload, assuming the frequency and magnitude of transient overload are bounded (and the bounds are trusted, akin to WCETs). Both techniques [45, 63] support arbitrary arrival curves.

Exceedance analysis, like all the alternatives cited above, is purely a *design-time analysis* that allows exploring "what-if scenarios." It does not have a runtime component itself, but it can be readily combined with a wide range of classic *runtime mechanisms* that prevent or contain overruns. This includes RTOS *budget enforcement* [*e.g.*, 29, 46, 51, 53, 56], more sophisticated *reservation schemes* [*e.g.*, 6, 17, 51, 65], *time-partitioning* [*e.g.*, 25, 41], and *job abortion* [*e.g.*, 57].

If *all* NETs are *precisely* enforced, then exceedance analysis is not necessary. Nonetheless, it remains useful and relevant in hybrid setups with partial NET enforcement, including reservations or budgets shared among multiple tasks [*e.g.*, 26, 47, 53, 56], real-time virtualization [*e.g.*, 7, 10, 71], intra-partition schedulers [*e.g.*, 55, 68], job-abortion thresholds larger than NETs, or if not all tasks are subject to enforcement (*e.g.*, interrupt service routines, SCHED_FIFO tasks in Linux, *etc.*).

Case in point, a recent survey of industrial practices reports that the use of reservations is far from ubiquitous [8, Q19], that measured NETs are pervasive [8, Q18], and that applications are commonly allowed to run past deadline misses [8, Q17]. This is not surprising as many real-time systems continue to be deployed on RTOSs (or RTOS standards) lacking reservations such as FreeRTOS [3], ThreadX [5], or AUTOSAR Classic [1].

X. Conclusion

We have considered the problem of response-time analysis in the absence of reliable WCET bounds. Our solution is to explicitly account for transient overloads that can occur when tasks temporarily exceed their nominal (*i.e.*, assumed) execution-time bounds. The resulting *exceedance analysis*—which is applicable to the FP, EDF, and FIFO scheduling policies, as well as to the full spectrum of preemption models—provides a *rigorous* and *systematic* means for assessing a system's *temporal safety margins* (w.r.t. overruns at runtime).

To aid engineers in exploring the space of exceedance effects, we have developed a fast search algorithm for enumerating response-time nonlinearities. Additionally, we have proposed a configurable trace generator that explains by means of example, in a manner that does not require a background in scheduling theory, how exceedance effects can affect response times. An empirical evaluation has shown both the search algorithm and the trace-generation method to be effective.

Future work should focus on exceedance analysis for other commonly measured task parameters such as self-suspension and critical-section lengths, release-jitter bounds, and generally uncertain arrival curves. It will also be interesting to extend our analysis to allow for processor-supply restrictions (*e.g.*, within shared reservations) and uncertain supply-bound functions.

Acknowledgements

We thank the reviewers for their insightful and constructive feedback, which has helped to significantly improve the paper.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 803111). This work has also been partially supported by the European Union's Horizon Europe framework programme project NANCY under the grant agreement No. 101096456, and the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

References

[1] "AUTOSAR Classic Platform: AUTOSAR's solution for embedded systems with hard real-time and safety constraints," https://www.autosar.org/standards/classic-platform.

[2] "The Coq proof assistant, project web site," https://coq.inria.fr.

[3] "FreeRTOS: Real-time operating system for microcontrollers and small microprocessors," https://freertos.org.

[4] "Prosa: The proven schedulability analysis repository," http://prosa.mpi-sws.org.

[5] "Eclipse ThreadX: Open source RTOS certified for safety-critical applications," https://threadx.io.

[6] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, 1998.

[7] L. Abeni and D. Faggioli, "Using Xen and KVM as real-time hypervisors," *Journal of Systems Architecture*, 2020.

[8] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "An empirical survey-based study into industry practice in real-time systems," in *Proceedings of the 41st IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[9] B. Andersson, "The case for explainability of real-time systems and their analyses," in *Proceedings of the 1st International Workshop on Explainability of Real-time Systems and their Analysis (ERSA)*, 2022.

[10] A. Avanzini, P. Valente, D. Faggioli, and P. Gai, "Integrating Linux and the real-time ERIKA OS through the Xen hypervisor," in *Proceedings of the 10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2015.

[11] P. Balbastre, I. Ripoll, and A. Crespo, "Analysis of window-constrained execution time systems," *Real-Time Systems*, 2007.

[12] S. Baruah and P. Ekberg, "Towards efficient explainability of schedulability properties in real-time systems," in *Proceedings of the 35th Euromicro Conference on Real-Time Systems (ECRTS)*, 2023.

[13] K. Bedarkar, M. Vardishvili, S. Bozhko, M. Maida, and B. B. Brandenburg, "From intuition to Coq: A case study in verified response-time analysis of FIFO scheduling," in *Proceedings of the 43rd IEEE Real-Time Systems Symposium (RTSS)*, 2022.

[14] E. Bini, M. Di Natale, and G. Buttazzo, "Sensitivity analysis for fixed-priority real-time systems," *Real-Time Systems*, 2008.

[15] L. Bougueroua, L. George, and S. Midonnet, "Temporal robustness of real-time architectures specified by estimated WCETs," *International Journal On Advances in Software*, 2009.

[16] S. Bozhko and B. B. Brandenburg, "Abstract response-time analysis: A formal foundation for the busy-window principle," in *Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS)*, 2020.

[17] S. A. Brandt, S. Banachowski, C. Lin, and T. Bisson, "Dynamic integrated scheduling of hard real-time, soft real-time, and non-real-time processes," in *Proceedings of the 24th IEEE Real-Time Systems Symposium (RTSS)*, 2003.

[18] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, 2009.

[19] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. A survey," *IEEE Transactions on Industrial Informatics*, 2013.

[20] G. C. Buttazzo, "Rate monotonic vs. EDF: Judgment day," *Real-Time Systems*, 2005.

[21] M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, D. L. Woodruff *et al.*, *Pyomo-optimization modeling in Python*, 2021.

[22] F. Cerqueira, F. Stutz, and B. B. Brandenburg, "Prosa: A case for readable mechanized schedulability analysis," in *Proceedings of the 28th Euromicro Conference on Real-Time Systems (ECRTS)*, 2016.

[23] J. Chen, C. Du, P. Han, and Y. Zhang, "Sensitivity analysis of strictly periodic tasks in multi-core real-time systems," *IEEE Access*, 2019.

[24] J. Chen, Z. Feng, J.-Y. Wen, B. Liu, and L. Sha, "A container-based DoS attack-resilient control framework for real-time UAV systems," in *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[25] A. Crespo, I. Ripoll, and M. Masmano, "Partitioned embedded architecture based on hypervisor: The XtratuM approach," in *Proceedings of the 5th European Dependable Computing Conference (EDCC)*, 2010.

[26] D. Dasari, M. Becker, D. Casini, and T. Blaß, "End-to-end analysis of event chains under the QNX adaptive partitioning scheduler," in *Proceedings of the 28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022.

[27] F. Dorin, P. Richard, M. Richard, and J. Goossens, "Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities," *Real-Time Systems*, 2010.

[28] P. Fradet, M. Lesourd, J.-F. Monin, and S. Quinton, "A generic Coq proof of typical worst-case analysis," in *Proceedings of the 39th IEEE Real-Time Systems Symposium (RTSS)*, 2018.

[29] P. K. Gadepalli, R. Gifford, L. Baier, M. Kelly, and G. Parmer, "Temporal capabilities: Access control for time," in *Proceedings of the 38th IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[30] M. K. Gardner and J. W.-S. Liu, "Performance of algorithms for scheduling real-time systems with overrun and overload," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS)*, 1999.

[31] L. George and J.-F. Hermant, "Characterization of the space of feasible worst-case execution times for earliest-deadline-first scheduling," *Journal of Aerospace Computing, Information, and Communication*, 2009.

[32] W. Giernacki, P. Kozierski, J. Michalski, M. Retinger, R. Madonski, and P. Campoy, "Bebop 2 quadrotor as a platform for research and education in robotics and control engineering," in *Proceedings of the 2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2020.

[33] D. Griffin, I. Bate, and R. I. Davis, "Generating utilization vectors for the systematic evaluation of schedulability tests," in *Proceedings of the 41st IEEE Real-Time Systems Symposium (RTSS)*, 2020.

[34] Z. Guo, S. Vaidhun, A. Al Arafat, N. Guan, and K. Yang, "Stealing static slack via WCRT and sporadic p-servers in deadline-driven scheduling," in *Proceedings of the 44th IEEE Real-Time Systems Symposium (RTSS)*, 2023.

[35] A. Hamann, D. Dasar, S. Kramer, M. Pressler, F. Wurst, and D. Ziegenbein, "WATERS industrial challenge 2017," in *Proceedings of the 8th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.

[36] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Extending typical worst-case analysis using response-time dependencies to bound deadline misses," in *Proceedings of the 14th International Conference on Embedded Software (EMSOFT)*, 2014.

[37] Z. A. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux, "Bounding deadline misses in weakly-hard real-time systems with task dependencies," in *Proceedings of the 2017 Design,*

Automation & Test in Europe Conference & Exhibition (DATE), 2017.

[38] D. Hardy, B. Rouxel, and I. Puaut, "The Heptane static worst-case execution time estimation tool," in *Proceedings of the 17th International Workshop on Worst-Case Execution Time Analysis (WCET)*, 2017.

[39] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: Modeling and solving mathematical programs in Python," *Mathematical Programming Computation*, 2011.

[40] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the SymTA/S approach," *IEE Proceedings-Computers and Digital Techniques*, 2005.

[41] P. Karachatzis, J. Ruh, and S. S. Craciunas, "An evaluation of time-triggered scheduling in the Linux kernel," in *Proceedings of the 31st International Conference on Real-Time Networks and Systems (RTNS)*, 2023.

[42] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, 2018.

[43] T. Kessler, J. Bernhard, M. Buechel, K. Esterle, P. Hart, D. Malovetz, M. T. Le, F. Diehl, T. Brunner, and A. Knoll, "Bridging the gap between open source software and vehicle hardware for autonomous driving," in *Proceedings of the 30th IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[44] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *Proceedings of the 6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[45] P. Kumar and L. Thiele, "Quantifying the effect of rare timing events with settling-time and overshoot," in *Proceedings of the 33rd IEEE Real-Time Systems Symposium (RTSS)*, 2012.

[46] A. Lackorzyński, A. Warg, M. Völp, and H. Härtig, "Flattening hierarchical scheduling," in *Proceedings of the 10th ACM International Conference on Embedded Software (EMSOFT)*, 2012.

[47] C. Lee, R. Rajkumar, and C. Mercer, "Experiences with processor reservation and dynamic QoS in real-time Mach," in *Proceedings of Multimedia Japan*, 1996.

[48] J. Lee, J. Wang, D. Crandall, S. Šabanović, and G. Fox, "Real-time, cloud-based object detection for unmanned aerial vehicles," in *Proceedings of the 1st IEEE International Conference on Robotic Computing (IRC)*, 2017.

[49] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proceedings of the 10th Real-Time Systems Symposium (RTSS)*, 1989.

[50] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of the 11th Real-Time Systems Symposium (RTSS)*, 1990.

[51] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Software: Practice and Experience*, 2016.

[52] H. Leppinen, "Current use of Linux in spacecraft flight software," *IEEE Aerospace and Electronic Systems Magazine*, 2017.

[53] A. Lyons, K. McLeod, H. Almatary, and G. Heiser, "Scheduling-context capabilities: A principled, light-weight operating-system mechanism for managing time," in *Proceedings of the 13th EuroSys Conference*, 2018.

[54] M. Maida, S. Bozhko, and B. B. Brandenburg, "Foundational response-time analysis as explainable evidence of timeliness," in *Proceedings of the 34th Euromicro Conference on Real-Time Systems (ECRTS)*, 2022.

[55] J. Martins and S. Pinto, "Shedding light on static partitioning hypervisors for Arm-based mixed-criticality systems," in *Proceedings of the 29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2023.

[56] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: An abstraction for managing processor usage," in *Proceedings of the 4th IEEE Workshop on Workstation Operating Systems (WWOS-III)*, 1993.

[57] S. Natarajan, M. Nasri, D. Broman, B. B. Brandenburg, and G. Nelissen, "From code to weakly hard constraints: A pragmatic end-to-end toolchain for Timed C," in *Proceedings of the 40th IEEE Real-Time Systems Symposium (RTSS)*, 2019.

[58] P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *Proceedings of the 40th IEEE Real-Time Systems Symposium (RTSS)*, 2019.

[59] S. Punnekkat, R. Davis, and A. Burns, "Sensitivity analysis of real-time task sets," in *Advances in Computing Science — ASIAN'97: Proceedings of the Third Asian Computing Science Conference*, 1997.

[60] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *Proceedings of the 2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.

[61] S. Quinton, M. Negrean, and R. Ernst, "Formal analysis of sporadic bursts in real-time systems," in *Proceedings of the 2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2013.

[62] R. Racu, A. Hamann, and R. Ernst, "A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems," in *Proceedings of the 18th Euromicro Conference on Real-Time Systems (ECRTS)*, 2006.

[63] ——, "Sensitivity analysis of complex embedded real-time systems," *Real-Time Systems*, 2008.

[64] S. Ranjha, G. Nelissen, and M. Nasri, "Partial-order reduction for schedule-abstraction-based response-time analyses of non-preemptive tasks," in *Proceedings of the 28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022.

[65] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Real-Time Systems*, 1989.

[66] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2011.

[67] Tesla Motors, "Linux distribution for Tesla vehicles," https://github.com/teslamotors/linux, 2024.

[68] M. Vanga, A. Bastoni, H. Theiling, and B. B. Brandenburg, "Supporting low-latency, low-criticality tasks in a certified mixed-criticality OS," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems (RTNS)*, 2017.

[69] S. Vestal, "Fixed-priority sensitivity analysis for linear compute time models," *IEEE Transactions on Software Engineering*, 1994.

[70] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, 2008.

[71] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: Towards real-time hypervisor scheduling in Xen," in *Proceedings of the 9th ACM International Conference on Embedded Software (EMSOFT)*, 2011.

[72] W. Xu, Z. A. Hammadeh, A. Kröller, R. Ernst, and S. Quinton, "Improved deadline miss models for real-time systems using typical worst-case analysis," in *Proceedings of the 27th Euromicro Conference on Real-Time Systems (ECRTS)*, 2015.

[73] R. Yerraballi, R. Mukkamala, K. Maly, and H. Wahab, "Issues in schedulability analysis of real-time systems," in *Proceedings of the 7th Euromicro Workshop on Real-Time Systems (ECRTS)*, 1995.

[74] F. Zhang, A. Burns, and S. Baruah, "Sensitivity analysis for EDF scheduled arbitrary deadline real-time systems," in *Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTAS)*, 2010.