Improving and Estimating the Precision of Bounds on the Worst-Case Latency of Task Chains

Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia, Nicolas Sordon

Improving and Estimating the Precision of Bounds on the Worst-Case Latency of Task Chains

Alain Girault, Christophe Prévot and Sophie Quinton
Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
Grenoble, France
firstname.lastname@inria.fr

Rafik Henia and Nicolas Sordon
Thales Research and Technology
Palaiseau, France
firstname.lastname@thalesgroup.com

ABSTRACT

One major issue that hinders the use of performance analysis in industrial design processes is the pessimism inherent to any analysis technique that applies to realistic system models. Indeed, such analyses may conservatively declare unschedulable systems that will in fact never miss any deadlines. We advocate the need to compute not only tight upper bounds on worst-case behaviors, but also tight lower bounds. As a first step, we focus on uniprocessor systems executing a set of sporadic or periodic hard real-time task chains. Each task has its own priority, and the chains are scheduled according to the fixed-priority preemptive scheduling policy. Computing the worst-case end-to-end latency (WCEL) of each chain is complex because of the intricate relationship between the task priorities. Compared to the state of the art, our analysis provides upper bounds on the WCEL in the more general case of asynchronous task chains, and also provides lower bounds on the WCEL both for synchronous and asynchronous chains. Our computed lower bounds correspond to actual system executions exhibiting a behavior that is as close to the worst case as possible, while all other approaches rely on simulations. Extensive experiments show the relevance of lower bounds on the worst-case behavior for the industrial design of real-time embedded systems.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**;

KEYWORDS

Worst-case end to end latency, Latency analysis, Task chains

ACM Reference format:
Alain Girault, Christophe Prévot and Sophie Quinton and Rafik Henia and Nicolas Sordon. 2018. Improving and Estimating the Precision of Bounds on the Worst-Case Latency of Task Chains. In *Proceedings of EMSOFT, Torino, Italy, September 2018 (EMSOFT'18),* 11 pages.
https://doi.org/

EMSOFT'18, September 2018, Torino, Italy

ACM ISBN . . . $15.00
https://doi.org/

1 INTRODUCTION

Timing is crucial for the correct execution of hard real-time systems with strict requirements on worst-case end-to-end latencies (WCEL). Verifying these timing requirements becomes more and more challenging due to the increasing complexity of systems. Over approximations are thus mandatory, resulting in pessimistic upper bounds on WCEL. The problem is that the end user has no idea about how pessimistic such upper bounds are. We therefore advocate the need to compute also a *lower bound* on WCEL[1], computed from execution scenarios guaranteed to be feasible. Such lower bounds can of course also be obtained by simulating the system with thousands of execution scenarios and keeping the largest value. Our experiments show that, in general, the lower bounds obtained through simulation are lower than our computed ones.

In this paper, we apply this principle to hard real-time systems consisting of *chains* of tasks executed on a single core processor under the Fixed-Priority Preemptive (FPP) policy. Our task chains follow a periodic or a sporadic activation model, have arbitrary deadlines and can be synchronous or asynchronous. This system model is common to many industrial systems, e.g. *On-Board Software* (OBSW) in satellites or *Flight Management Systems* (FMS) in avionics: OBSW and FMS are both uniprocessor systems executing periodic and sporadic task chains. The FMS is scheduled according to the ARINC653 standard [1] that defines time partitions in which tasks are scheduled under FPP. For both systems, the strict certification constraints (DO178C for avionics [3] and ECSS-E-ST-40C for space [2]) impose demonstrating the correctness of the timing behavior.

When tasks are independent and scheduled with FPP, computing the worst-case response time of each task is a well understood problem because the interference that each task may be subject to is limited to its higher-priority tasks. In the case of task chains however, the problem is a lot more complex: A given task chain σ will be subject to the interference of any other task chain that contains at least one task of a priority higher than the lowest priority of the tasks in σ.

Recent work on the analysis of task chains [15] proposes a solution to this problem, but with significant over-approximation. Earlier work provides upper bounds [8] for synchronous chains but is restricted to synchronous chains. We propose here a novel solution that is both tighter than [15] and that applies to *synchronous and asynchronous* task chains. We define a priority order on chains that allows us to reason about latency analysis in a way that is similar to the response-time analysis of [7] for Fixed-Priority Non-Preemptive scheduling of independent tasks (FPNP). Finally,

[1] A lower bound on the WCEL must not be confused with the best-case end-to-end latency (BCEL).

and most importantly, we are able to compute also *lower bounds* on the WCEL of task chains. Interestingly, computing lower bounds turns out to be a much more complex problem for periodic task chains than for sporadic ones. Based on the computed lower bounds, we can estimate the precision of the computed upper bound on the latency of each task chain.

The paper is organized as follows. Section 2 discusses related work and how our analysis improves over it. Section 3 introduces our system model. Sections 4, 5, 6, and 7 formalize our approach to compute upper bounds on the latency of task chains. Section 8 develops our method to compute lower bounds on task chain latencies. Section 9 provides extensive experiments that show the usefulness of the combined use of upper and lower bounds on worst-case latencies. Section 10 concludes and discusses future work.

2 RELATED WORK

There exists a huge body of literature dealing with the real-time scheduling of tasks and the computation of worst-case response times and latencies. We focus in this section on the particular case of tasks with precedence constraints.

The two papers that are most closely related to our paper are [8] and [15]. Although [8] uses a different terminology (namely, tasks and subtasks instead of chains of tasks), the underlying model used in these two papers and ours is identical. Moreover, both papers address the problem of computing the WCEL of task chains on a single-core processor under the FPP scheduling policy, but they only focus on providing upper bounds and do not discuss at all the tightness of their bounds.

In [8], Gonzalez Harbour et al. propose a framework to analyze the schedulability of a real-time system consisting of a set of periodic synchronous tasks, where each task is itself a sequence of subtasks. They introduce a canonical form where consecutive subtasks have increasing priorities, and they prove that the latency of the task under study remains the same if it is put under canonical form. We improve their analysis in three directions. (1) We present a very precise formalization and we *formally prove* its correctness ([8] provides no formal proof of correctness). (2) Our analysis applies both to synchronous and to *asynchronous chains with arbitrary deadlines*, i.e., chains with self-interference of forthcoming instances (while Assumption 3 in [8] excludes this case). (3) And we compute also *lower bounds* on the WCEL.

In [15], Schlatow and Ernst extend the Compositional Performance Analysis (CPA) of [9] to chains of tasks. Compared to CPA, this reduces significantly the pessimism of the computed upper bounds on latencies. Still, the drawback of [15] is to use the same definition for the two distinct concepts of busy window and q-event busy time (see Section 4), which hinders the comprehension of the underlying mechanisms of the analysis. As we demonstrate in our paper, this incurs a significant pessimism. Compared to [15], we greatly improve the WCEL analysis by computing tighter upper bounds and by providing also lower bounds (which allow the tightness of the WCEL to be measured).

As presented in [17] and then extended to more complex systems in [13], offsets may be used to model precedence constraints: tasks are grouped into transactions such that tasks of the same transaction do not interfere with each other. Offset-based latency analysis, which builds on top of task response time analysis, improves over standard latency analysis without dependencies. Still, [15] shows through experiments that the analysis in [15] (over which we improve) outperforms offset-based analysis.

There is a body of research on parallel applications, where tasks are split into subtasks with precedence constraints that form a graph, in particular the fork-join model [11], the synchronous parallel task model [14] and the DAG-based task model [4]. Corresponding analyses thus address more complicated systems and their computed upper bounds are very conservative. We have found no contributions presenting a formal analysis of lower bounds for such systems. It would indeed be interesting to provide such an analysis.

In this paper, we focus on functional task chains, where the end of a task activate the upcoming one, in contrast to cause-effect chains as in [5], where the dependencies between tasks are data dependencies. In a cause-effect chain, each task is activated independently but reads data produced by the previous task in the chain before executing. The systems we target ultimately are multiprocessor with functional chains on a processor and data dependencies between processors. In future work, we plan to extend our analysis to handle jointly functional and data dependencies.

3 SYSTEM MODEL

Unless otherwise specified, all the parameters defined in the following have positive integer values. In particular, we assume a *discrete time clock*. We consider a *uniprocessor* real-time system S consisting of a finite set of m *task chains* scheduled with the FPP scheduling policy. All task chains are *independent*, meaning that two chains cannot share a task and there is neither task fork nor task join.

Definition 3.1 (Task). A *task* τ_a^i is defined as a pair (π_a^i, C_a^i) with π_a^i the priority and C_a^i the worst-case execution time (WCET) of τ_a^i.

Definition 3.2 (Task chain). A *task chain* $\sigma_a \in S$, $a \in \{1, \ldots, m\}$, is defined by:

- A finite sequence of n_a distinct tasks denoted $(\tau_a^1, \tau_a^2, \ldots, \tau_a^{n_a})$ with precedence relations such that, for each $i \in \{1, \ldots, n_a - 1\}$, τ_a^{i+1} is activated at the completion time of τ_a^i.
- An activation model (see Defs. 3.3 and 3.4) that specifies the activation instants of the first task in the chain τ_a^1.
- A relative deadline D_a (see Def. 3.5 below).
- A synchronous or asynchronous execution policy (see Def. 3.6).

All priorities are assumed to be *distinct*. We use the convention that $\pi_a^i > \pi_b^j$ means that τ_a^i has a higher priority than τ_b^j. As a result, τ_a^i may preempt τ_b^j when it arrives.

Task chains are activated from external sources, which can be either periodic timers or various types of sensor devices. We model the activation patterns of chains using *arrival functions*, or their pseudoinverses called *distance functions*. These functions can be used to model sporadic as well as periodic activations [9].

Definition 3.3 (Arrival function). A *maximum (resp. minimum) arrival function* $\eta_a^+ : \mathbb{N} \to \mathbb{N}$ (resp. η_a^-) returns, for any time interval Δ, an upper (resp. lower) bound on the number of activations of chain σ_a that can arrive within any time interval $[t, t + \Delta[$.

Definition 3.4 (Distance function). A *minimum (resp. maximum) distance function* $\delta_a^- : \mathbb{N} \to \mathbb{N}$ (resp. δ_a^+) returns, for any $q \in \mathbb{N}$, a

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

lower (resp. upper) bound on the length of any time interval that contains q activations of chain σ_a.

Without loss of generality, in the rest of this paper we use the above notations for both periodic and sporadic chains. Note that sporadic chains do not have maximum distance functions: two sporadic activations can be arbitrarily far apart. In contrast, periodic activations are exactly separated by a period.

Definition 3.5 (Deadline). We distinguish two types of relative deadlines for a chain σ_a: *Constrained* deadline: $D_a \leq \delta_a^-(2)$. *Arbitrary* deadline: no constraint on D_a.

The timing behavior of a task chain σ_a is an infinite sequence of *instances*, each of them made of one instance of each task in the chain, such that: (i) the arrival time of τ_a^1 is defined by η_a^+; (ii) the arrival time of τ_a^{i+1} is the completion time of τ_a^i. Task chains may behave differently in presence of multiple instances [15].

Definition 3.6 (Synchronous, asynchronous chain). We distinguish:

- *Synchronous* chains: an instance of chain σ_a cannot start execution until previous instances of σ_a have completed.
- *Asynchronous* chains: an instance of chain σ_a may preempt previous instances of σ_a.

Instances of an asynchronous chain σ_a can thus suffer from interferences due to (i) other chains; (ii) previous instances of σ_a; and (iii) later instances of σ_a. A synchronous chain will only suffer from (i) and (ii). This distinction is irrelevant if all deadlines are constrained, as two instances of a chain cannot overlap in that case (unless there is a deadline miss). We now introduce a few notations.

Notation 1. *Given a chain σ_a, the sequence of tasks $(\tau_a^p, \ldots, \tau_a^m)$, $1 \leq p \leq m \leq n_a$, is denoted $\sigma_{a[p..m]}$ and called a* subchain.

Notation 2. *Given a chain σ_a, C_a denotes $\sum_{i=1}^{n_a} C_a^i$, the execution time of σ_a. We extend this notation to all subchains, $C_{a[p..m]}$ denoting the execution time of $\sigma_{a[p..m]}$.*

Priorities can be in *any order*, not necessarily ascending or descending. The following example provides some intuition regarding the complexity of the resulting timing behavior of task chains.

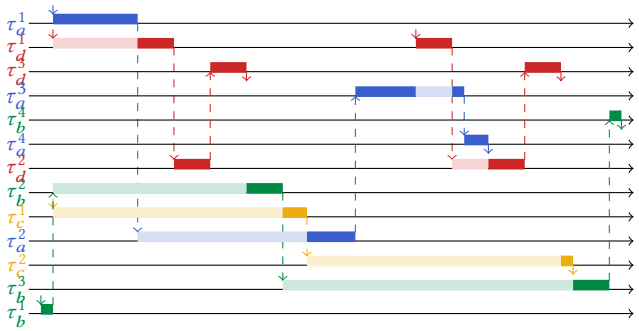

Figure 1: A system with four task chains. Task priorities are decreasing from top to bottom.

Example 3.7. Fig. 1 shows an execution of a system with four task chains that interfere according to complex patterns because their priorities are interleaved. For example, task τ_a^2 has a priority lower than τ_b^2, so the activation of τ_a^3 is indirectly delayed by the lower-priority task τ_b^2. Handling such interferences is the main challenge when computing the WCEL of task chains.

Definition 3.8 (End-to-end latency). The end-to-end *latency* of an instance of a task chain σ_a is the duration between the arrival of τ_a^1 and the completion time of $\tau_a^{n_a}$ of the same instance.

The *worst-case end-to-end latency* (WCEL) ℓ_a of σ_a is the maximum latency over all possible instances of σ_a.

Definition 3.9 (Schedulable). A system is *schedulable* if and only if all instances of all chains are guaranteed to meet their deadline.

Notation 3. *For any value V, $\overline{V}$ and $\underline{V}$ denote respectively an upper and a lower bound on V.*

The schedulability of a real-time system $S = \{\sigma_a\}_{a=1}^m$ is usually assessed by computing, for each chain σ_a, an *upper bound* $\overline{\ell_a}$ on its WCEL, and checking that $\overline{\ell_a} \leq D_a$. Computing a widely over-estimated $\overline{\ell_a}$ results in many systems being declared as non schedulable. The problem we address in this paper is therefore twofold. First, we provide a framework to compute upper bounds that are as tight as possible given the complexity of the analysis. Second, we compute, for each chain, a scenario (chosen to be close to the worst case) that exhibits a *realizable* value for its latency. This value constitutes a *lower bound* on its WCEL. We can thus measure the pessimism of our WCEL analysis.

4 UPPER BOUNDS ON CHAIN LATENCIES

In this section, we develop the main concepts needed for computing upper bounds on task chain latencies. We start with an observation: Any chain σ_a has n_a different priority levels (remember that n_a denotes the number of tasks in σ_a), but for most of our WCEL analysis, we only need to consider the lowest priority task of each chain. These notations can also be applied to subchains. Note that neither [8] nor [15] use chain priorities, which makes their developments much harder to read.

Definition 4.1 (Priority of a chain). The *priority* of task chain σ_a, denoted π_a, is the priority of its lowest priority task:

$$\pi_a = \min_{i=1..n_a} \{\pi_a^i\}$$

Task τ_b^j has a lower priority than a chain σ_a if and only if $\pi_b^j < \pi_a$.

Since all task priorities are different, chain priorities define a *total order* over task chains.

Notation 4. *We use $\ell p(a)$, resp. $hp(a)$, to denote the set of chains with a strictly lower, resp. strictly higher, priority than σ_a. Also, $hpe(a) = hp(a) \cup \{\sigma_a\}$. We denote $\ell p_b(a)$ the set of tasks of σ_b that have lower priority than σ_a, i.e., $\ell p_b(a) = \{\tau_b^j \in \sigma_b | \pi_b^j < \pi_a\}$.*

4.1 Upper bounds on busy windows

Most response time analyses for uniprocessor systems rely on some notion of busy window (or busy period) and this paper is no exception. We use here the same notion as in [8], which extends the original concept of [12] to task chains.

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

Definition 4.2 (σ_a-busy window). A *σ_a-busy window* is a maximal time interval during which there is always (at least) one instance of a task with priority higher than or equal to σ_a that is pending, i.e., it has been previously activated but has not finished yet.

In particular, a σ_a-busy window cannot be closed until all pending instances of σ_a and higher-priority *chains* have finished their execution. Among lower-priority chains, only tasks with a priority higher than σ_a are considered as part of a σ_a-busy window.

Example 4.3. Our running example of Fig. 1 shows two σ_a-busy windows: the first one starts with the activation of σ_a and ends with the completion of the second instance of σ_d. The second one spans the execution of τ_b^4.

We will see that, similar to busy-window approaches such as [7], task instances (of any chain) can only interfere with instances of σ_a that are in the same σ_a-busy-window. It is therefore useful to have an upper bound on the length of a σ_a-busy-window.

Definition 4.4 (Lower-priority interference). We call *lower-priority interference* and denote $\ell p\mathcal{I}_a(\Delta)$ the maximum amount of time that chains with priority lower than σ_a may execute in any prefix of length Δ of a σ_a-busy window.

The function $\ell p\mathcal{I}_a$ can be used to provide an upper bound on the amount of time σ_a may be delayed by lower-priority chains inside a σ_a-busy window. We will show in Section 6.1 how to compute such an interference.

Theorem 4.5. *Let σ_a be a task chain. The length of any σ_a-busy window is upper bounded by the least fixed point $\overline{BW_a}$ of Eq.* (1)*:*

$$\overline{BW_a} = \ell p\mathcal{I}_a(\overline{BW_a}) + \sum_{\sigma_b \in hpe(a)} \eta_b^+(\overline{BW_a}) \times C_b \tag{1}$$

Proof. By definition, lower-priority chains execute for at most $\ell p\mathcal{I}_a(\overline{BW_a})$ in any prefix of a σ_a-busy window of length $\overline{BW_a}$. Besides, there cannot be any instance of σ_a or of a higher-priority chain pending at the beginning of a σ_a-busy-window; and any such instance that is activated within a σ_a-busy-window is by definition guaranteed to fully execute before the end of that σ_a-busy-window. Each chain $\sigma_b \in hpe(a)$ therefore accounts for C_b times its maximal number of instances activated within $\overline{BW_a}$, that is, $\eta_b^+(\overline{BW_a})$. □

Property 1. *Let σ_a be a task chain. The number of activations of σ_a in a σ_a-busy-window is upper bounded by $\overline{K_a} = \eta_a^+(\overline{BW_a})$.*

Proof. The proof follows directly from Theorem 4.5. □

The above results still hold if an upper bound is used for the lower-priority interference. In contrast, proving that the upper bound on the length of a σ_a-busy-window is reachable (i.e., there exists a σ_a-busy-window not greater than that length) requires to prove that the maximum lower-priority interference and the maximum interference from higher-priority chains can be achieved in one single execution scenario (see Section 8).

4.2 Upper bounds on busy times

In order to upper bound the latency of σ_a, and similar to e.g. [7], we need to first focus on the time it may take to finish executing q instances of σ_a within a σ_a-busy-window, for $q \in [1, \overline{K_a}]$. The following definition is the adaptation to task chains of the concept with the same name introduced in [16]. A σ_a-busy-window does not necessarily close when a σ_a instance finishes executin because there may be, e.g., pending instances of higher-priority chains. This implies that, although two instances of σ_a cannot overlap in a schedulable system with constrained deadlines, they may still be part of the same σ_a-busy-window.

Definition 4.6 (q-event busy time). The *q-event busy time* of a chain σ_a (resp. a task τ_a^i), denoted $B_a(q)$ (resp. $B_a^i(q)$), is the maximum time duration it may take to finish processing the first q instances of σ_a (resp. τ_a^i) within any σ_a-busy-window that contains at least q instances of σ_a.

To upper bound the q-event busy time of σ_a for $q \in \{1, \ldots, \overline{K_a}\}$, we will upper bound the q-event busy time of some tasks in σ_a, depending their priority w.r.t. the priority of the chains in $hp(a)$.

Theorem 4.7. *The q-event event busy time of chain σ_a is equal to the q-event event busy time of its last task: $B_a(q) = B_a^{n_a}(q)$.*

Proof. This directly follows from the definitions. □

We can thus focus on upper bounding the q-event event busy time of the tasks in σ_a. For that, we distinguish between the interference due to lower- and higher-priority chains, as well as possible interference from subsequent activations of σ_a.

Definition 4.8 (q-event interference). The *q-event lower-priority (resp. higher-priority) (resp. self) interference* wrt a task τ_a^i, denoted $\ell p\mathcal{I}_a^{i,q}(\Delta)$ (resp. $hp\mathcal{I}_{b\to a}^{i,q}(\Delta)$) (resp. $self\mathcal{I}_a^{i,q}(\Delta)$) is the maximum amount of time that chains with priority lower than σ_a (resp. a chain σ_b with $\pi_a < \pi_b$) (resp. forthcoming instances of σ_a) may execute in any prefix of length Δ of a σ_a-busy window that finishes at the end of the q-th execution of τ_a^i.

Remember that if deadlines are constrained, or for synchronous chains, self-interference cannot happen.

Property 2. *The q-event busy time of τ_a^i is upper bounded by*

$$\overline{B_a^i(q)} = (q-1) \times C_a + C_{a[1..i]} + \ell p\mathcal{I}_a^{i,q}(\overline{B_a^i(q)}) + \sum_{\sigma_b \in hp(a)} hp\mathcal{I}_{b\to a}^{i,q}(\overline{B_a^i(q)}) + self\mathcal{I}_a^{i,q}(\overline{B_a^i(q)}) \tag{2}$$

Proof. The maximum time it may take to fully process the first q instances of τ_a^i within a σ_a-busy-window is upper bounded by the sum of the maximum time it takes to: (1) compute σ_a entirely $q-1$ times; (2) compute the q-th instance of σ_a until task τ_a^i (i.e., the WCET of the subchain $\sigma_{a[1..i]}$); (3) account for the interference due to (i) chains with lower priority than σ_a; (ii) chains with higher priority than σ_a; (iii) subsequent activations of σ_a. □

This results holds also if one or several of the interference delays is upper bounded. To upper bound such q-event busy times, one needs to upper bound the corresponding interference. We will show in the next sections how to achieve this. Before that, let us show

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

how upper bounds on q-event busy times are used to upper bound the latency of task chains.

4.3 Upper bounds on latencies

Once the busy times are upper bounded, upper bounds on the WCEL of task chains are easily obtained.

Theorem 4.9 (Worst-case latency). *The WCEL of task chain σ_a is bounded by*

$$\overline{\ell_a} = \max_{q \in [1, \overline{K_a}]} \{B_a(q) - \delta_a^-(q)\} \quad (3)$$

Proof. Consider any instance σ_a^x of σ_a. As a consequence of the definition of σ_a-busy-window, σ_a^x is part of a (unique) σ_a-busy-window. Thanks to Prop. 1, we know that there exists $q \in [1, \overline{K_a}]$ such that σ_a^x is the q-th instance in its σ_a-busy-window. It follows directly from the definition of the q-event busy time that σ_a^x cannot finish later than $B_a(q)$ after the beginning of the σ_a-busy-window. Besides, σ_a^x cannot be activated earlier than $\delta_a^-(q)$ after the beginning of the σ_a-busy-window. Hence the result. □

The above result stands even if $B_a(q)$ is upper bounded. The next sections focus on upper bounding the lower-priority, higher-priority and self-interference.

5 SEGMENTS

Like lower-priority tasks in [7], lower-priority chains in our context may interfere in complex ways, as we discuss at the end of this section. To discuss lower- and higher-priority interference, we develop (and formalize) the concept of segment introduced in [15].

Intuitively, a *segment* of a chain σ_b w.r.t. a chain σ_a such that $\pi_b < \pi_a$ is a maximal subchain of σ_b that may delay σ_a. The task immediately before or immediately after a segment of σ_b w.r.t. σ_a has lower priority than σ_a, i.e., the task belongs to $\ell p_b(a)$.

Definition 5.1 (Inner segment $s_{b\to a}$). An *inner segment* of σ_b wrt σ_a is a subchain $(\tau_b^i, \ldots, \tau_b^j)$ of σ_b s.t. $1 < i \leq j < n_b$, $\tau_b^{i-1}, \tau_b^{j+1} \in \ell p_b(a)$ and $\forall k \in i..j$, $\tau_b^k \notin \ell p_b(a)$.

Definition 5.2 (Head segment $s_{b\to a}^{head}$). If $\tau_b^1 \notin \ell p_b(a)$, $(\tau_b^1, \ldots, \tau_b^{i-1})$ is the *head segment* of σ_b wrt σ_a, where τ_b^i is the first task in σ_b that is in $\ell p_b(a)$.

Definition 5.3 (Tail segment $s_{b\to a}^{tail}$). If $\tau_b^{n_b} \notin \ell p_b(a)$, $(\tau_b^{j+1}, \ldots, \tau_b^{n_b})$ is the *tail segment* of σ_b wrt σ_a, where τ_b^j is the last task in σ_b that is in $\ell p_b(a)$.

Definition 5.4 (Circular segment $s_{b\to a}^{circ}$). If $\tau_b^1 \notin \ell p_b(a)$ and $\tau_b^{n_b} \notin \ell p_b(a)$ then the *circular segment* of σ_b wrt σ_a is the concatenation of the tail and the head segment of σ_b wrt σ_a.

Definition 5.5 (Critical segment $s_{b\to a}^{crit}$). The *critical segment* of σ_b wrt σ_a is the segment of σ_b wrt σ_a with the longest execution time.

Example 5.6. In our running example of Fig. 1, σ_b has wrt σ_a one inner segment (τ_b^2), no head segment, one tail segment (τ_b^4), no circular segment and its critical segment is (τ_b^2).

Intuitively, the critical segment of σ_b wrt σ_a is the segment of σ_b that can interfere the most with σ_a. We can now state one key property regarding segments, which will allow us to compute precisely the interference of lower-priority chains. This result does not appear in [15], which leads to pessimistic results. In contrast, this result is used in [8], but without a formal proof, and only in the synchronous case.

Property 3. *Let σ_a, σ_b, and σ_c be three task chains such that $\pi_a > \pi_b > \pi_c$. Any two segments $s_{b\to a}$ and $s_{c\to a}$ that are not head segments cannot execute (and therefore delay σ_a) within the* same *σ_a-busy window.*

Proof. The proof is by contradiction. Consider a σ_a-busy window in which both segments $s_{b\to a}$ and $s_{c\to a}$ execute. Let τ_b^k (resp. τ_c^j) be the task that precedes $s_{b\to a}$ (resp. $s_{c\to a}$) in σ_b (resp. σ_c). According to Def. 5.1, $\tau_b^k \in \ell p_b(a)$ and $\tau_c^j \in \ell p_b(a)$.

τ_b^k and τ_c^j have a lower priority than all tasks in σ_a and therefore also a lower priority than all tasks in chains with higher priority than σ_a. As a result, neither τ_b^k nor τ_c^j can execute within the considered σ_a-busy window (which "closes" before they can execute).

$s_{b\to a}$ and $s_{c\to a}$ execute within this busy window so τ_b^k and τ_c^j must execute *before* the considered σ_a-busy window.

Let us assume that τ_b^k finishes executing before τ_c^j and thus activates $s_{b\to a}$. Remember that tasks in $s_{b\to a}$ and $s_{c\to a}$ have a higher priority than π_a, while τ_b^k and τ_c^j have a lower priority than π_a. As a result, $s_{b\to a}$ will execute before τ_c^j can execute. This is a contradiction since τ_c^j must execute before the considered σ_a-busy window and $s_{b\to a}$ within that same window. The same argument applies to the case where τ_c^k finishes executing before τ_b^j. □

The above result does not make any assumption wrt deadlines (constrained or not) or execution policy (synchronous or asynchronous). Other results are however simpler if constrained deadlines can be assumed. For that reason, we first present the constrained deadline case before providing the general results in Section 7.

6 THE CONSTRAINED DEADLINE CASE

In this section, we assume that deadlines are constrained (see Def. 3.5) s.t. the distinction between synchronous and asynchronous chains is irrelevant (since two instances of the same chain cannot be pending at the same time). We provide formulas for lower-priority as well as higher-priority interference. The general case where deadlines are arbitrary is dealt with in the next section.

6.1 Lower-priority interference

A first key property on segments that is used to bound the interference from lower priority task chains on σ_a is that *only one* segment per lower priority chain may interfere in any σ_a-busy-window.

Property 4. *Suppose that deadlines are constrained. Let σ_a and σ_b be task chains s.t. $\pi_b < \pi_a$. In any σ_a-busy-window, σ_b executes at most one segment, possibly circular.*

Proof. A task between two segments of σ_b is such that $\pi_b^k < \pi_a$, so after executing a segment of σ_b, the task that follows the segment will be preempted until the end of the σ_a-busy-window. □

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

Prop. 4 only holds for constrained deadlines. If deadlines are arbitrary, for asynchronous chains, header segments of later instances can also execute. Prop. 4 allows us to bound by a constant the interference incurred on chain σ_a by its lower priority chains, supposing that deadlines are constrained.

Theorem 6.1. *Suppose that all deadlines are constrained and let σ_a be a chain. In any σ_a-busy window, the set of chains with a lower priority than σ_a execute for at most*

$$\overline{\ell p \mathcal{I}_a} = \max_{\sigma_b \in \ell p(a)} \left\{ C_{s^{crit}_{b\to a}} + \sum_{\sigma_c \in \ell p(a) \wedge c \neq b} C_{s^{head}_{c\to a}} \right\} \quad (4)$$

Proof. According to Prop. 4, a chain with lower priority than σ_a can execute in a σ_a-busy window at most one segment. According to Prop. 3, no two chains with lower priority than σ_a can execute a non-head segment in any σ_a-busy window. It follows that the largest interference due to the $|\ell p(a)|$ lower-priority task chains is the maximum among all combinations of 1 critical segment and $|\ell p(a)|-1$ head segments, which is formalized by Eq. (4). □

Example 6.2. In our running example, Fig. 1 shows the worst-case lower-priority interference on chain σ_a (from σ_b and σ_c).

6.2 Higher-priority interference

In [15], higher-priority chains are conservatively assumed to interfere for their entire execution time when they are activated during the execution of a chain σ_a. In fact, the exact interference of higher-priority chains is more complex and we can provide tighter bounds than this, as illustrated in the following example.

Example 6.3. Refer to Fig. 1, and consider the interference of σ_d on σ_a. Even if $\pi_a < \pi_d$, the second activation of σ_d cannot arrive before part of σ_a has finished executing, and it will thus only partially interfere with σ_a.

We exploit this observation in the following to propose tighter bounds on higher-priority interference. Throughout this section, we assume given two chains σ_a and σ_b s.t. $\pi_a < \pi_b$.

Notation 5. *For a given chain $\sigma_b \in hp(a)$, we denote by $\tau_{\ell t_a(b)}$ the last task of σ_a that has lower priority than σ_b. We denote $\tau_{\ell t_a}$ the last task of chain σ_a that has lower priority than all chains in $hp(a)$.*

The following properties state that if all activations of σ_b within a prefix $[t_1, t_2]$ of a σ_a-busy-window arrive before the q-th instance of $\tau_a^{\ell t_a(b)}$ is guaranteed to have finished, then we consider that they may execute entirely before σ_a may complete its q-th instance. If σ_b is activated after that, then its interference can only be partial.

Property 5. *If $\ell t_a \leq i \leq \ell t_a(b)$ then the q-event interference of σ_b on τ_a^i can be bounded by*

$$\overline{hp\mathcal{I}^{i,q}_{b\to a}}(\Delta) = \eta_b^+(\Delta) \times C_b \quad (5)$$

Proof. Let $[t_1, t_2]$ be a time interval of length Δ that starts at the same time as a σ_a-busy window and finishes at the end of the q-th execution of τ_a^i, as in the definition of interference. By definition there are no activations of σ_b pending at t_1. In addition, no more than $\eta_b^+(\Delta)$ activations of σ_b may arrive within $[t_1, t_2]$, which cannot execute for longer than C_b each. Hence the result. □

Property 6. *If $i > \ell t_a(b)$ and deadlines are constrained, the q-event interference of σ_b on τ_a^i for $\Delta > B_a^{i-1}(q)$ is bounded by:*

$$\overline{hp\mathcal{I}^{i,q}_{b\to a}}(\Delta) = \eta_b^+(B_a^{\ell t_a(b)}(q)) \times C_b + I^{i,q}_{a,b} \quad (6)$$

$$\text{where} \quad I^{i,q}_{a,b} = \begin{cases} 0 & \text{if } \eta_b^+(B_a^{\ell t_a(b)}(q)) = \eta_b^+(\Delta) \\ C_{s^{head}_{b\to a[k..i]}} & \text{otherwise} \end{cases}$$

$$\text{and} \quad k = \min\left(i, \min_{k \in \{\ell t_a(b)+1, \ldots, i-1\}} \{\eta_b^+(B_a^{k-1}(q)) \neq \eta_b^+(B_a^k(q))\}\right)$$

Proof. If $\eta_b^+(B_a^{\ell t_a(b)}(q)) = \eta_b^+(\Delta)$ then the proof proceeds as for Prop. 5. Let us now suppose that $\eta_b^+(B_a^{\ell t_a(b)}(q)) < \eta_b^+(\Delta)$. Let $[t_1, t_2]$ be a prefix of length Δ of a σ_a-busy window that finishes at the end of the q-th execution of τ_a^i, as in the definition of interference. The proof proceeds as follows:

- **All activations of σ_b that arrive within $[t_1, t_1 + B_a^{\ell t_a(b)}(q)[$ may fully execute.**
- **The first activation of σ_b after $t_1 + B_a^{\ell t_a(b)}(q)$ may execute for up to $C_{s^{head}_{b\to a[k..i]}}$**: Let us consider the first activation of σ_b after $t_1 + B_a^{\ell t_a(b)}(q)$. By definition of k, it cannot arrive before $t_1 + B_a^{k-1}(q)$ but it may arrive before $t_1 + B_a^k(q)$ resp. $t_1 + \Delta$. That is, τ_a^{k-1} is guaranteed to have finished its q-th execution before that activation of σ_b, which means that τ_a^k is guaranteed to have been activated before it. The task of σ_b that is just after the segment $s^{head}_{b\to a[k..i]}$ has, by definition, a lower priority than all tasks in the subchain $\sigma_a[k..i]$. It will therefore not execute before the end of the q-th execution of τ_a^i, i.e., before $t_1 + \Delta = t_2$.
- **If σ_b is schedulable, then there are no other activations of σ_b that may arrive before t_2**: As we have just shown, the first activation of σ_b after $t_1 + B_a^{\ell t_a(b)}(q)$ cannot finish its execution before t_2. Hence the result if deadlines are constrained.

□

Prop. 6 implicitly entails an order in which busy times should be computed:

(1) Compute an upper bound on lower-priority interference $\overline{\ell p \mathcal{I}_a}$ as in Eq. (4).

(2) Compute an upper bound on the longest σ_a-busy-window using Eq. (1) and derive from it an upper bound on the maximum number $\overline{K_a}$ of activations of σ_a in any σ_a-busy-window as in Prop. 1.

(3) For $q \in \{1, \ldots, \overline{K_a}\}$, compute $\overline{B_a(q)}$ as follows:

- Compute $B_a^{\ell t_a}(q)$ by using Eq. (6) inside of Eq. (2). This involves a fixed point iteration which can be initialized with $(q-1) \times C_a + C_{a[1..i]} + \ell p \mathcal{I}_a$, as usual.
- For $i \in \{\ell t_a + 1, \ldots, n_a\}$, initialize the fixed point iteration with $\overline{B_a^{i-1}(q)} + C_i$, and then iteratively compute $\overline{B_a^i(q)}$ by using Eq. (6) inside Eq. (2).

Example 6.4. In our running example, Fig. 1 shows the worst-case higher-priority interference on chain σ_a. Note that the second instance of σ_d does not entirely interfere with σ_a.

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

7 THE ARBITRARY DEADLINE CASE

When deadlines are arbitrary, interferences depend on the nature of chains, i.e., whether they are synchronous or asynchronous. In this section, we first discuss our upper bound on higher-priority interference, which is a fairly simple generalization of our upper bound on higher-priority interference for constrained deadlines.

7.1 Higher-priority interference

Let us show how arbitrary deadlines affect our upper bound on higher-priority interference.

Theorem 7.1. *If σ_b is synchronous then the q-event interference of σ_b on τ_a^i can be upper bounded using the same bound as for constrained deadlines, i.e.: $\overline{hp\mathcal{I}^{i,q}_{\substack{b\to a\\ sync}}} = \overline{hp\mathcal{I}^{i,q}_{b\to a}}$.*

Proof. The only part of the correctness proof for $\overline{hp\mathcal{I}^{i,q}_{b\to a}}$ that uses the constrained deadline hypothesis is to justify that only one activation of σ_b that arrives after the completion of $\tau_a^{\ell t_a}(q)$ may interfere with σ_a. We also show that this interference is only partial, i.e., σ_b cannot complete this execution before the end of the q-th instance of τ_a^i. If σ_b is synchronous, an activation cannot start executing before the previous instance has completed. The same conclusion as for constrained deadlines thus follows. □

Theorem 7.2. *If $i > \ell t_a(b)$ and σ_b is asynchronous, the q-event interference of σ_b on τ_a^i for $\Delta > B_a^{i-1}(q)$ can be upper bounded by:*

$$\overline{hp\mathcal{I}^{i,q}_{\substack{b\to a\\ async}}}(\Delta) = \eta_b^+(B_a^{\ell t_a(b)}(q)) \times C_b + I^{i,q}_{a,b} \tag{7}$$

where

$$I^{i,q}_{a,b} = \sum_{\substack{k\in\{\ell t_a(b)+1,\ldots,i-1\}\\ n=\eta_b^+(B_a^k(q))-\eta_b^+(B_a^{k-1}(q))}} n \times C_{s^{head}_{b\to a[k..i]}} + \left(\eta_b^+(\Delta) - \eta_b^+\left(B_a^{i-1}(q)\right)\right) \times C_{s^{head}_{b\to a[i..i]}}$$

Proof. The difference with Prop. 6 is that now *all* activations of σ_b that arrive after the completion of $\tau_a^{\ell t_a(b)}$ may interfere, for a duration obtained as in Eq. (6). □

7.2 Lower-priority interference

The bound on lower-priority interference that is given in Eq. (4) for the constrained deadline case can be easily adapted to take into account the fact that asynchronous lower-priority chains can interfere for more than one header based on their arrival function.

Theorem 7.3. *The lower-priority interference on a chain σ_a in any prefix of length Δ of a σ_a-busy window is bounded by:*

$$\overline{lp\mathcal{I}_a}(\Delta) = \max_{\sigma_b\in lp(a)} \left\{ C_{s^*_{b\to a}} + \sum_{\sigma_c\in\ell p(a)\cap\mathcal{SC}\wedge c\neq b} C_{s^{head}_{c\to a}} + \sum_{\sigma_c\in\ell p(a)\cap\mathcal{AC}} \eta_c^+(\Delta)\times C_{s^{head}_{c\to a}} \right\} \tag{8}$$

where

$$s^*_{b\to a} = \begin{cases} s^{crit}_{b\to a} & \text{if the critical segment is not circular}\\ s^{crit}_{b\to a} & \text{if the critical segment is circular and } \sigma_b\in\mathcal{SC}\\ s^{tail}_{b\to a} & \text{otherwise}\end{cases}$$

Proof. This is the same formula as in Eq. (4), except that asynchronous chains can execute several headers. Note that if the critical segment is circular for an asynchronous chain σ_b, then its header part is already included in the last term of the sum. □

Following the same reasoning as the one presented for higher-priority interference, we can refine this upper bound by noticing that activations of asynchronous chains that arrive within the interval may not be able to execute their full header. We therefore use the finer-grained notion of q-event lower-priority interference, as opposed to the lower-priority interference used above, which considers interference at the chain level instead of the task level.

Theorem 7.4. *The q-event lower-priority interference wrt task τ_a^i for $\Delta > B_a^{i-1}(q)$ is upper bounded by:*

$$\overline{\ell p\mathcal{I}_a^{i,q}}(\Delta) = \max_{\sigma_b\in\ell p(a)} \left\{ C_{s^*_{b\to a}} + \sum_{\sigma_c\in\ell p(a)\cap\mathcal{SC}\wedge c\neq b} C_{s^{head}_{c\to a}} + \sum_{\sigma_c\in\ell p(a)\cap\mathcal{AC}} hp\mathcal{I}^{i,q}_{\substack{s^{head}_{c\to a}\to a\\ async}}(\Delta) \right\} \tag{9}$$

Proof. As in Eq. (8), only one chain in $hp(a)$ may execute an inner segment and all other chains execute their header. For asynchronous chains, several headers may interfere. The header segment $s^{head}_{c\to a}$ of an asynchronous, lower-priority chain σ_c interferes with σ_a exactly like a higher-priority chain does. □

7.3 Self-interference

In the case of asynchronous chains, self-interference $self\mathcal{I}_a^{i,q}(\Delta)$ is the interference of the header of σ_a on σ_a itself. This is the same principle as the header interference of lower priority chains on σ_a. Note that two instances of the same task τ_a^i may be pending at the same time. In this case, we apply a FIFO policy.

Theorem 7.5. *If σ_a is asynchronous, the q-event self-interference of σ_a on τ_a^i for $i \le \ell t_a$ and $\Delta > B_a^{i-1}(q)$, denoted $self\mathcal{I}_a^{i,q}(\Delta)$, can be upper bounded by:*

$$\overline{self\,\mathcal{I}_a^{i,q}}(\Delta) = \left(\eta_a^+(B_a^{\ell t_a}(q)) - q\right) \times C_{s^{head}_{a\to a[1..\ell t_a]}} + I_a^{i,q} \tag{10}$$

where $I_a^{i,q} = \left(\eta_a^+(\Delta) - \eta_a^+(B_a^{i-1}(q))\right) \times C_{s^{head}_{a\to a[i..i]}} + \sum_{\substack{k\in\{\ell t_a+1,\ldots,i-1\}\\ n=\eta_a^+(B_a^k(q))-\eta_a^+(B_a^{k-1}(q))}} n \times C_{s^{head}_{a\to a[k..i]}}$

Proof. All activations of σ_a, after the q-th one, that may arrive before $\tau_a^{\ell t_a}$ has completed, can interference up to the header of σ_a on itself. Subsequent activations can only interfere less, depending on how early they may arrive and which task are guaranteed to have completed by then. The reasoning is similar to previous proofs. □

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

8 PRECISION OF THE ANALYSIS

We now focus on quantifying the precision of the upper bound on the latency. For that, we discuss now how to compute, for each chain σ_a, a *lower bound* $\underline{\ell_a}$ on its WCEL — not to be confused with its BCEL: $\underline{\ell_a}$ expresses that there exists an actual system execution[2] in which the observed WCEL of σ_a is equal or larger than $\underline{\ell_a}$. If $\underline{\ell_a}$ and $\overline{\ell_a}$ are close, this means that the computed upper bound is a good approximation of the worst-case behavior of the system. For simplicity, we assume first that all deadlines are constrained.

Property 7. *Consider a σ_a-busy-window $[t_1, t_2]$ in which all chains in $hp(a)$ are activated together at t_1, then again as early as permitted by their activation model, and all task executions take their worst-case execution time to complete. Suppose that lower bounds $\underline{\tilde{B}_a^j(q)}$ are given on the actual finishing time of tasks for $j < i$. Then the higher-priority interference of σ_b on σ_a for $\Delta > \underline{\tilde{B}_a^{i-1}(q)}$ is at least $\overline{hp\mathcal{I}_{b\to a}^{i,q}}(\Delta)$, where $\overline{hp\mathcal{I}_{b\to a}^{i,q}}(\Delta)$ is computed using $\underline{\tilde{B}_a^j(q)}$ instead of $B_a^j(q)$ for $j < i$ in Eq. (6).*

Proof. **1.** The result holds for $\ell t_a \le i \le \ell t_a(b)$ based on Prop. 5: chain σ_b is in our scenario activated exactly $\eta^+(\Delta)$ times between t_1 and $t_1 + \Delta$ and since all these activations arrive before $\tau_a^{\ell t_a(b)}$ has completed, they execute entirely before the q-th instance of τ_a^i completes.

2. Consider $i > \ell t_a(b)$.

- If $\eta^+(\Delta) = \eta^+(\underline{\tilde{B}_a^{\ell t_a(b)}(q)})$, the above argument still holds.
- The remaining case is $\eta_b^+(\underline{\tilde{B}_a^{\ell t_a(b)}(q)}) < \eta_b^+(\Delta)$. Let k be as in Eq. (6). Because we are now working with lower bounds on the actual completion time of tasks, τ_a^k is the first task in σ_a that *cannot* finish its q-th execution before the first activation of σ_b after $t_1 + \underline{\tilde{B}_a^{\ell t_a(b)}(q)}$. That instance interferes for at least $C_{s_{b\to a[k..i]}^{head}}$.

□

Prop. 7 indicates a strategy for finding lower bounds on worst-case latencies, assuming that there is a way to lower bound lower-priority interference.

Corollary 8.1. *If there exists a σ_a-busy-window as in the above property such that the q-event interference of lower-priority chains is lower bounded by $\underline{\ell p\mathcal{I}_a^{i,q}}$ then*

$$\underline{B_a^i(q)} = (q-1) \times C_a + C_{a[1..i]} + \underline{\ell p\mathcal{I}_a^{i,q}}\left(\underline{B_a^i(q)}\right) + \sum_{\sigma_b \in hp(a)} \overline{hp\mathcal{I}_{b\to a}^{i,q}}\left(\underline{B_a^i(q)}\right) \quad (11)$$

is a lower bound on the q-event busy time of task τ_a^i.

Proof. Assume that σ_a executes for its worst-case execution time. It cannot complete its q-th instance of τ_a^i before $t_1 + \underline{B_a^i(q)}$ as it suffers at least $\underline{\ell p\mathcal{I}_a^{i,q}}(\underline{B_a^i(q)})$ interference from lower-priority chains as well as at least $\overline{hp\mathcal{I}_{b\to a}^{i,q}}(\underline{B_a^i(q)})$ from higher-priority chains (from Prop. 7). □

[2]Note that by actual execution, we mean an execution that is allowed by the model. Of course the model is an abstraction of the real system, so such an execution may still not be possible on the real platform.

This result naturally extends to lower bounds on latencies if we further assume that chain σ_a is activated at t_1 and then again as early as permitted by its activation model. To compute these lower bounds, as higher priority interference is exact, we need to provide a lower bound $\underline{\ell p\mathcal{I}_a^{i,q}}(\underline{B_a^i(q)})$ on lower priority interference.

The rest of this section will be is devoted to computing $\overline{\ell p\mathcal{I}_a^{i,q}}$. The principle is to exhibit a scenario that is guaranteed to be feasible, and to compute the lower bound $\underline{\ell p\mathcal{I}_a^{i,q}}$ from this scenario. We propose in fact to take the maximum over several scenarios, resulting in several lower bounds $\underline{\ell p\mathcal{I}_a}_{(n)}^{i,q}$:

$$\underline{\ell p\mathcal{I}_a}^{i,q} = \max_n \underline{\ell p\mathcal{I}_a}_{(n)}^{i,q} \quad (12)$$

8.1 All task chains are sporadic

Let us start by assuming that all task chains are sporadic — and remember that we assume so far that deadlines are constrained. Indeed, the sporadic case is the simpler one for computing lower bounds. In order to use Corollary 8.1 we must provide a lower bound for lower-priority interference for σ_a-busy-windows $[t_1, t_2]$ with the properties required in Corollary 8.1, in particular those from Prop. 7, i.e.:

- Chain σ_a is activated at t_1, then again as early as permitted by its activation model (such that the result extends to lower bounds on latencies).
- The same holds for all chains in $hp(a)$ (for Prop. 7 to hold).
- All task executions in chains of $hpe(a)$ take their worst-case execution time to complete.

This leaves only room for specifying the activation scenario of lower-priority chains. To do so, we distinguish two cases, presented in the following paragraphs.

8.1.1 The critical segment maximizing Eq. (4) is not circular. This is the most intuitive scenario.

Theorem 8.2. *Suppose that the critical segment that maximizes Eq. (4) is not circular. Then Eq. (4) is also a lower bound on the worst-case lower-priority interference of chain σ_a under the above conditions.*

Proof. Denote σ_{crit} the chain such that it's critical segment maximizes $\overline{\ell p\mathcal{I}_a}$ in Eq. (4) and assume that this critical segment is not circular. Consider now the following activation scenario for lower-priority chains. All chains in $\ell p(a)$ are activated at t_1, except chain σ_{crit} that is activated at $t_1 - C_{crit[1..\xi_{crit}]}$, where ξ_{crit} is the index of the task that precedes the critical segment. We further assume that there are no other activation of any chain before t_1, which is a feasible scenario since all chains are sporadic, and that all chains in $\ell p(a)$ always run for their worst-case execution time.

In such a scenario, the header of each chain in $\ell p(a)\backslash\{\sigma_{crit}\}$ is activated at t_1, as well as the critical segment of σ_{crit}. All tasks in these segments have higher priority than $\tau_a^{\ell t_a}$ and are thus guaranteed to execute within any prefix of $[t_1, t_2]$ that finishes with the completion of a task τ_a^i for $i > \ell t_a$. Hence the result. □

In this case the upper bound on the latency is reachable and thus the analysis is tight.

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

8.1.2 The critical segment maximizing Eq. (4) is circular. When the critical segment of σ_{crit} is circular, things are more complex. Indeed, there is an additional condition for this segment to be guaranteed to fully execute before the end of the σ_a-busy-window: the second instance of σ_{crit} must be activated before the end of $\tau_a^{\ell t_a(crit)}$. Otherwise, we are left with various options for:

- proving that the computed upper bound is not feasible and thus tightening it. This may be the case for example if $\delta_b^-(2)$ is large compared to $B_b^{crit}(1)$ and $B_a^{\ell t_a(crit)}(1)$. Unfortunately, only rather naive sufficient conditions will scale.
- finding other lower bounds on the WCEL.

In the following, we investigate the latter option.

Definition 8.3 (Non-circular critical segment $s_{b\to a}^{nocirc}$). The *non-circular critical segment* of σ_b w.r.t. σ_a, is the non-circular segment of σ_b w.r.t. σ_a that has the largest worst-case execution time.

Theorem 8.4. *When the critical segment is circular, a lower bound on the lower-priority interference on σ_a is:*

$$\underline{\ell p I_a}_{(1)} = \max_{\sigma_b \in \ell p(a)} \left\{ C_{s_{b\to a}^{nocirc}} + \sum_{\sigma_c \in \ell p(a) \wedge c \neq b} C_{s_{c\to a}^{head}} \right\} \tag{13}$$

Proof. The reasoning in the proof of Theorem 8.2 holds for any lower-priority chain, not only σ_{crit}. This theorem is thus a direct application of the same principle. □

In order to reduce the difference between lower and upper bound further, one could also investigate whether the proposed upper bound can be improved. This requires a much finer-grained analysis, as the worst-case lower-priority interference may not coincide with the worst-case higher-priority interference scenario we have been working with so far. Still, our experiments show that, in most cases, $\underline{\ell p I_a}_{(1)}$ is close to $\overline{\ell p I_a}$.

8.2 At least one task chain is periodic

Interestingly, the periodic case is the most complex one when computing lower bounds. The reason is that one cannot assume that there are no activations other than the one from σ_{crit} before t_1. Finding an alternative scenario that takes into account the constraints induced by the periodic activations is far from trivial. We therefore prefer to rely on the simpler scenario where all chains are activated at the same instant as σ_a, and hence all chains in $\ell p(a)$ interfere with σ_a with their head segment. In other words, there is no critical chain anymore. This scenario yields the following lower bound on the blocking time:

Theorem 8.5. *When at least one chain is periodic, a lower bound on the lower-priority interference on σ_a is:*

$$\underline{\ell p I_a}_{(2)} = \sum_{\sigma_b \in \ell p(a)} C_{s_{b\to a}^{head}} \tag{14}$$

Proof. The proof proceeds similarly to that of Theorem 8.2. □

8.3 Lifting the constrained deadline hypothesis

In this section, we briefly sketch how the results under the constrained deadline assumptions generalize to the general case of arbitrary deadlines.

- Prop. 7 and Corollary 8.1 can be directly adapted for higher-priority interference of chains with arbitrary deadlines by using the definitions of Section 7.
- Regarding low-priority interference, we have the same problem as in Section 8.1.2 for all σ_{crit} that are asynchronous, i.e., we do not know exactly how early a subsequent activation may arrive. Still, we can use the lower bound provided in Th. 8.5.

9 EXPERIMENTAL EVALUATION

We now describe how we generated our test cases and evaluated our latency analysis. Our analysis is implemented inside standalone Python tool. We have generated systems with at least one periodic and one sporadic chain. Utilization breakdowns are randomly chosen amongst the following values: $[0.4, 0.5, 0.6, 0.7]$ and the utilization breakdown of sporadic chains is chosen amongst: $[0.001, 0.01, 0.1]$ (utilization breakdown of periodic chains is the difference between both previous values). We randomly choose the number of chains between 2 and 9 and the number of periodic chains between 1 and 8 (there is at least one periodic task chain). We deduce the number of sporadic task chains. The number of tasks per chain is also a random value between 1 and 9. For periodic task chains, the periods are randomly chosen amongst $[10, 20, 50, 100, 200, 500, 1000]$. The above parameters have been chosen as they are considered for many industrial cases [10]. Chain and task utilizations are generated using U-Unifast [6]. The WCET of each chain is deduced using periodic utilization and periods. For sporadic chains, the WCET is a random value between 1 and 100 and the arrival function is defined s.t. it fits the sporadic utilization for $\delta^-(100)$ (which we consider large enough). Priorities of all tasks are randomly assigned. Altogether, the generated systems contain 5,538 chains. These chains have been analyzed first under the synchronous semantics (Fig. 2) and then under the asynchronous semantics (Fig. 3), except for those chains that took too long to analyze (which explains why Fig. 3 has only 4,148 chains).

We have evaluated two criteria: (i) first, we have compared our analysis to [15], which makes some approximations in the computation of both higher-priority and lower-priority interference; we did not compare our analysis with that of [8] because, for synchronous chains the two analyses coincide, and for asynchronous chains [8] does not work; (ii) second, we have evaluated the precision of our bounds as given by the computed lower bounds.

Figs. 2 and 3 report the experimental results respectively for synchronous and asynchronous chains. To each chain correspond two points in Figures 2 and 3: a red point that indicates our computed upper bound and a blue point showing the upper bound computed using the method proposed in RTAS 2016. In addition, in Fig 2, a green point showing our computed lower bound. For readability, all generated chains are sorted according to our computed upper bound on their WCEL. The exponential shape of our graphs results from our generation methods for sporadic chain activations, which may be bursty up to $\delta^-(100)$. Due to space limitations, Figures 2 and 3 illustrate our results only for the first group of systems. The trend is similar for both groups of systems, but the precision (i.e., the difference between upper and lower bound) is better for the sporadic case (not shown), as was to be expected from the theoretical results. For systems with periodic and sporadic chains, however,

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

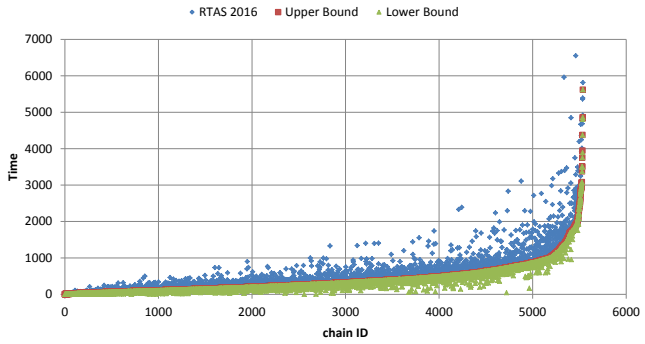

Figure 2: Latency bounds for periodic and sporadic synchronous chains.

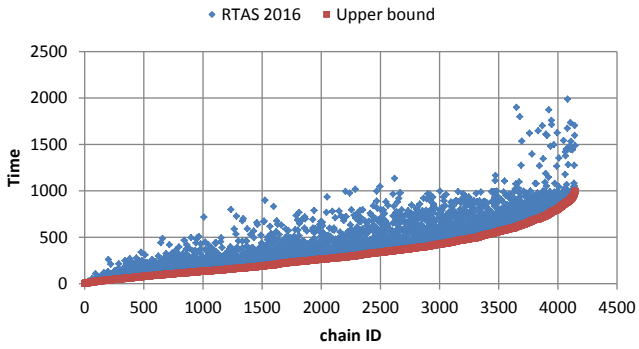

Figure 3: Latency bounds for periodic and sporadic asynchronous chains.

the relative difference between the lower and the upper bound can be large. Additional experiments have shown that the number of tasks per chain, the number of chains in the system, the length of the task chains, or the utilization breakdown do not significantly influence the precision of the obtained bounds.

Since lower bounds can also be obtained by *simulating* the system, we have simulated the system of Fig. 1 on $1,000,000$ randomly generated activation scenarios, and measured the simulated lower bound for each of its four chains. Table 1 summarizes the results. For σ_a and σ_b, the simulated lower bound is identical to the computed one. For σ_c and σ_d, the computed lower bound improves the simulated one respectively by 95% and 71%. Since these results were obtained for a single system, more experiments are required to compare the simulated and the computed lower bounds.

Fig. 4 depicts the evolution of the simulated lower bound in function of the number of activation scenarios. Table 2 reports an equivalent simulation but for a tight system consisting of six chains, again simulated over $1,000,000$ randomly generated activation scenarios. The lower bound obtained by simulation is sometimes tight (e.g., for chains σ_a and σ_e), but otherwise it can be very far from the computed lower bound (e.g., for chain σ_b). All this demonstrates the usefulness of our analysis. In both tables, the percentages are computed as:

$$\frac{\text{computed } \underline{\ell_i} - \text{simulated } \underline{\ell_i}}{\text{simulated } \underline{\ell_i}} \times 100$$

chain i	$\overline{\ell_i}$	computed $\underline{\ell_i}$	simulated $\underline{\ell_i}$	diff.
σ_a	24	16	16	0%
σ_b	35	34	34	0%
σ_c	42	41	21	95%
σ_d	48	48	28	71%

Table 1: Simulated and computed lower bounds for the system of Fig. 1, over $1,000,000$ activation scenarios.

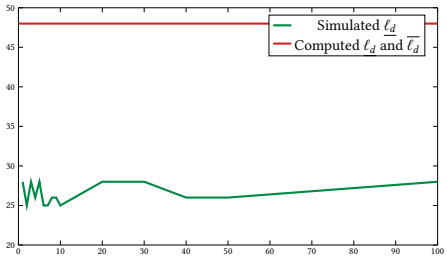

Figure 4: Evolution of the simulated lower bound $\underline{\ell_d}$.

chain i	$\overline{\ell_i}$	computed $\underline{\ell_i}$	simulated $\underline{\ell_i}$	diff.
σ_a	55	55	55	0%
σ_b	178	178	38	368%
σ_c	124	124	75	65%
σ_d	176	176	122	44%
σ_e	286	286	286	0%
σ_f	37	37	14	164%

Table 2: Simulated and computed lower bounds for a tight system over $1,000,000$ activation scenarios.

In summary, our experimental results show the following:

- Our upper bound analysis significantly improves over [15].
- In most cases, our lower bound analysis is able to guarantee that the computed upper bound is fairly tight.
- There are, however, quite a few instances for which upper and lower bounds differ significantly. This underlines the value of such an information.
- The lower bounds obtained by simulations are sometimes significantly less than the ones computed by our analysis.

10 CONCLUSION

In this paper, we propose an improved performance analysis technique allowing the computation of tighter upper bounds for task chain latencies in uniprocessor systems compared to the state of the art, and providing an innovative approach to assess the quality

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.

of the computed bounds by comparing them to lower worst-case bounds from feasible execution scenarios. We also present a set of experiments that show the gain obtained in term of analysis precision when using our solution.

We believe that our analysis represents an important step toward the acceptance of performance analysis techniques in the industrial design process of real-time embedded systems. One should take notice that a major reason hindering the use of performance analysis in the industry is not only the over-dimensioning induced by the various approximations used in current analyses, but also the lack of methods to quantify it.

Future work will extend our solution to make it applicable to more complex industrial real-time systems by adding offsets, equal priorities and support for multiprocessor systems. In addition, the analysis technique we propose in this paper also represents an important step toward the computation of task chain latencies in multiprocessor systems. Very often in industrial multiprocessor systems, e.g., in software defined radios, after finishing its execution, a task will only activate the next task in the chain in case both are mapped to the same processor. When they are mapped to different processors, the activation of the next task is instead independent from the termination of the first task: the first task writes its output data in a memory, which are then read by the next task upon activation. The computation of the WCEL for such task chains will require using our analysis technique to compute the latency of the sub-chains on each processor, combined with a mechanism to analyze cause-effect chains between processors.

REFERENCES

[1] 2003. ARINC Specification 653-1. Avionics application standard interface. Aeronautical Radio Inc Software. (october 2003).

[2] 2009. ECSS-E-ST-40C. (March 2009). http://ecss.nl/standard/ecss-e-st-40c-software-general-requirements/

[3] 2010. DO178C Software Considerations in Airborne Systems and Equipment Certification. (may 2010). http://dx.doi.org/10.1049/ip-cdt:20045088

[4] Sanjoy K. Baruah, Vincenzo Bonifaci, Alberto Marchetti-Spaccamela, Leen Stougie, and Andreas Wiese. 2012. A Generalized Parallel Task Model for Recurrent Real-time Processes. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*. 63–72. https://doi.org/10.1109/RTSS.2012.59

[5] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2017. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture - Embedded Systems Design* 80 (2017), 104–113. https://doi.org/10.1016/j.sysarc.2017.09.004

[6] Enrico Bini and Giorgio C. Buttazzo. 2005. Measuring the Performance of Schedulability Tests. *Real-Time Systems* 30, 1-2 (2005), 129–154. https://doi.org/10.1007/s11241-005-0507-9

[7] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. 2007. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems* 35, 3 (2007), 239–272. https://doi.org/10.1007/s11241-007-9012-7

[8] M. González Harbour, M.H. Klein, and J.P. Lehoczky. 1991. Fixed Priority Scheduling Periodic Tasks with Varying Execution Priority. In *Real-Time Systems Symposium, RTSS'91*. IEEE, 116–128. https://doi.org/10.1109/REAL.1991.160365

[9] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. 2005. System Level Performance Analysis — the SymTA/S Approach. *IEEE Proceedings Computers and Digital Techniques* 152, 2 (April 2005), 148–166.

[10] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. 2015. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. https://www.ecrts.org/forum/viewtopic.php?f=20&t=23

[11] Karthik Lakshmanan, Shinpei Kato, and Ragunathan Rajkumar. 2010. Scheduling Parallel Real-Time Tasks on Multi-core Processors. In *Proceedings of the 31st IEEE Real-Time Systems Symposium, RTSS 2010, San Diego, California, USA, November 30 - December 3, 2010*. 259–268. https://doi.org/10.1109/RTSS.2010.42

[12] J.P. Lehoczky. 1990. Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines. In *Real-Time Systems Symposium, RTSS'90*. IEEE, Lake Buena Vista (FL), USA, 201–209. https://doi.org/10.1109/REAL.1990.128748

[13] J.C. Palencia and M. González Harbour. 1998. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Real-Time Systems Symposium, RTSS'98*. IEEE, Madrid, Spain, 23–37. https://doi.org/10.1109/REAL.1998.739728

[14] Abusayeed Saifullah, Kunal Agrawal, Chenyang Lu, and Christopher D. Gill. 2011. Multi-core Real-Time Scheduling for Generalized Parallel Task Models. In *Proceedings of the 32nd IEEE Real-Time Systems Symposium, RTSS 2011, Vienna, Austria, November 29 - December 2, 2011*. 217–226. https://doi.org/10.1109/RTSS.2011.27

[15] Johannes Schlatow and Rolf Ernst. 2016. Response-Time Analysis for Task Chains in Communicating Threads. In *Real-Time Embedded Technology and Applications Symposium, RTAS'16*. IEEE, 245–254. https://doi.org/10.1109/RTAS.2016.7461359

[16] Simon Schliecker. 2011. *Performance Analysis of Multiprocessor Real-Time Systems with Shared Resources*. PhD Thesis. TU Braunschweig, Braunschweig, Germany. http://goo.gl/AwxhDS

[17] Ken Tindell. 1994. Adding Time-Offsets to Schedulability Analysis. (1994), 28. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.8815&rep=rep1&type=pdf

This article was presented in the International Conference on Embedded Software 2018 and appears as part of the ESWEEK-TCAD special issue.