CertiCAN: Certifying CAN Analyses and Their Results

Pascal Fradet, Xiaojie Guo, Sophie Quinton

CertiCAN
Certifying CAN Analyses and Their Results

Pascal Fradet, Xiaojie Guo, Sophie Quinton

RESEARCH REPORT
N° 9443
Dec. 2021
Project-Team Spades

ISSN 0249-6399 ISRN INRIA/RR--9443--FR+ENG

CertiCAN
Certifying CAN Analyses and Their Results

Pascal Fradet*, Xiaojie Guo†*, Sophie Quinton*

Project-Team Spades

Research Report n° 9443 — Dec. 2021 — 32 pages

Abstract: We present CertiCAN, a tool produced using the Coq proof assistant for the formal certification of CAN analysis results. Result certification is a process that is lightweight and flexible compared to tool certification. Indeed, the certification of an industrial analyzer needs access to the source code, requires the proof of many optimizations or implementation tricks and new proof effort at each software update. In contrast, CertiCAN only relies on the result provided by such a tool and remains independent of the tool itself or its updates. Furthermore, it is usually more time efficient to check a result than to produce it. All these reasons make CertiCAN a practical choice for industrial purposes.
CertiCAN is based on the certification and combined use of two well-known CAN analysis techniques completed with additional optimizations. Experiments demonstrate that CertiCAN is computationally efficient and faster than the underlying combined analysis. It is able to certify the results of RTaW-Pegase, an industrial CAN analysis tool, even for large systems. This result paves the way for a broader acceptance of formal tools for the certification of real-time systems analysis results.

Key-words: Real-Time Systems Analysis, CAN bus, Certification, Coq Proof Assistant

* Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG
† Univ. Grenoble Alpes, CNRS, Grenoble INP, Verimag

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

CertiCAN
Certification d'analyses de bus CAN et de leurs résultats

Résumé : Nous présentons CertiCAN, un outil produit à l'aide de l'assistant de preuves Coq et permettant de certifier les résultats d'analyses de bus CAN. La certification de résultats d'analyses est un procédé moins coûteux et plus flexible que la certification des analyses elles-mêmes. En effet, la certification d'un analyseur industriel demande l'accès au code source, la preuve d'un code souvent complexe avec de nombreuses optimisations ou astuces de mise en œuvre sans compter le besoin de nouvelles preuves à chaque mise à jour. CertiCAN, de son côté, n'a besoin que du résultat de l'analyse ; il reste indépendant de l'analyseur, de son implantation et de ses mises à jour. De plus, il est toujours plus efficace de vérifier un résultat que de le produire. Ces caractéristiques font de CertiCAN un choix pragmatique pour une utilisation industrielle.

CertiCAN est basé sur la certification et l'utilisation combinée de deux techniques classiques d'analyse complétée par une collection d'optimisations. Les expérimentations montrent que CertiCAN est efficace et plus rapide que les analyses sous-jacentes. Il est capable de certifier les résultats de RTaW-Pegase, un analyseur industriel de bus CAN, et ce même pour des systèmes complexes. Ce résultat ouvre la voie à une utilisation plus large d'outils formels pour la certification des résultats d'analyse de systèmes temps-réels.

Mots-clés : Analyse de systèmes temps-réel, bus CAN, certification, assistant de preuves Coq

1 Introduction

There is a general trend toward certified[1] proofs for real-time systems analysis. One motivation is the need to increase confidence in the analysis techniques developed by the research community. A recent series of mistakes in the analysis of self-suspending tasks [8] underlines the limitations of pen-and-paper proofs for such complex problems. This issue is not new, as illustrated by the flaw in the original Response Time Analysis (RTA) of CAN messages proposed by Tindell *et al.* [26, 28, 27], which was found and fixed only many years later [11]. This motivated the development of Prosa [10], an open-source library of definitions and proofs for real-time systems analysis based on the Coq [4] proof assistant. Computer assisted proofs provide the additional advantage that they make it easier to build on top of existing results and to precisely identify the hypotheses required for a result to hold.

A second reason behind the need to certify real-time systems analysis results comes from industry. Standards such as ISO 26262 for automotive or DO-178C for avionics advocate the use of formal methods for the development and validation of safety critical systems. In particular, proof assistants have now reached a level of maturity which allows them to certify complex applications – see, for example, the use of the CompCert C compiler [3], [18] based on Coq or the Sel4 microkernel [7] based on Isabelle/HOL [5]. It is only natural that such a general trend towards formal proofs also affects real-time aspects. For all these reasons, we aim at providing certified real-time guarantees for industrial systems.

The analysis underlying our CertiCAN tool is a RTA for task sets with offsets under Fixed Priority Non Preemptive (FPNP) scheduling, with a notion of transaction, i.e., messages sent from the same Electronic Control Unit (ECU). The CAN [9] protocol is widely used in automotive applications and there exist several commercial tools performing CAN analysis. Among these, we focus on RTaW-Pegase [6], for which we obtained an academic license.

Rather than certifying RTaW-Pegase, that is, formally proving that the CAN analysis implemented in RTaW-Pegase is correct, we choose to build a tool based on the Coq proof assistant that can certify the results of the CAN analysis performed by RTaW-Pegase. In other words, our tool, called CertiCAN, can be called every time a result obtained with RTaW-Pegase[2] must be certified. This choice is motivated by the fact that result certification is a process that is lightweight and flexible compared to tool certification, which makes it a practical choice for industrial purposes. Indeed, RTaW-Pegase is a complex tool for which we do not have the source code. It is likely to be highly optimized and subject to regular changes. All this would make it difficult to certify the tool directly and this correctness proof would need to be updated regularly.

Our problem is then: Can we certify efficiently enough the analysis results computed by RTaW-Pegase? Compared to a traditional RTA, can we use the fact that a result certifier is given as input the bound it is expected to certify?

Our solution is based on the following idea: We use a combination of two existing analysis techniques, one precise but with high computational complexity and another one much faster but approximate (it may compute pessimistic upper bounds on response times). These two analyses were introduced by Tindell for the RTA of tasks with offsets scheduled according to the Fixed Priority Preemptive policy [24, 25]. The precise analysis was adapted to CAN by Meumeu Yomsi *et al.* [29].

CertiCAN combines in an optimized way the two analyses. It first tries to certify the result provided as input using the approximate analysis only, then resorts to a more precise analysis

[1] Throughout this paper, we use the term *certified* for *formally verified* using a proof assistant, in our case Coq.

[2] Note that CertiCAN does not depend on the internals of the RTA tool considered. It is interoperable with any other tool analyzing the same CAN system models as RTaW-Pegase.

for cases which cannot be certified using the approximate analysis. Experiments demonstrate the potential of CertiCAN for industrial practice. It is able to certify the results returned by RTaW-Pegase even for large systems.

The main contribution of this paper is CertiCAN, the first formally proven tool able to certify the results of commercial CAN analysis tools. This is however not the only contribution of the paper. More specifically, we propose:

1. a generic CAN analysis that permits to build concrete analyses with different precision, *e.g.,* precise, approximate, or a trade-off between them;
2. a new RTA for CAN that combines two well-known analyses, one precise and another approximate;
3. the correctness proof in Coq of these three analyses;
4. three Coq-certified tools in OCaml extracted from the proofs, one for each analysis;
5. based on the same principle as the new RTA, a method and its corresponding certified tool (called CertiCAN) for certifying the results of non certified tools such as RTaW-Pegase;
6. proven optimizations for increasing CertiCAN's timing efficiency;
7. experiments that demonstrate the usability of CertiCAN for industrial practice.

Beyond CertiCAN, we believe that the results presented in this paper are significant in that they demonstrate the usability of result certification for industrial analyzers. In addition, the underlying technique can be reused for any other system model for which there exist RTAs with different levels of precision.

All the Coq specifications and proofs are available online [2].

The rest of this paper is structured as follows. Section 2 introduces the system model and some notations and definitions used later on. We present in Section 3 the two existing CAN analyses as well as their generic version on which our certifier is based. Section 4 presents an optimized RTA combining the previous CAN analyses, which is then used in CertiCAN, a tool formally specified and proven in Coq for certifying CAN analysis results. Section 5 describes some proven optimizations needed to make CertiCAN deal with large systems. Section 6 relates experimental evaluations and comparisons of the combined analysis, CertiCAN and RTaW-Pegase. Additional details about the proof effort and the generality of the approach are provided in Section 7. Related work is presented in Section 8 and we conclude in Section 9.

This article extends and revises the work presented in [16, 14]. Since then, CertiCAN has been refined and greatly improved using novel (certified) optimizations. Sections 4.2, 5 and 7 are novel, new experiments have been conducted and Section 6 has been completely rewritten. Explanations and examples have been added throughout.

2 Context

The CAN network is a vehicle communication bus which is widely used in many industrial domains, especially in automotive. In critical applications, it is essential to perform RTAs to ensure that systems can meet their timing requirements (*e.g.,* that the response time of a message is smaller than its deadline). The CAN analyses offered by RTaW-Pegase are based on a precise RTA of periodic tasks with offsets dispatched according to the Fixed Priority Non-Preemptive (FPNP) scheduling policy [29]. In addition to the precise analysis, RTaW-Pegase proposes an

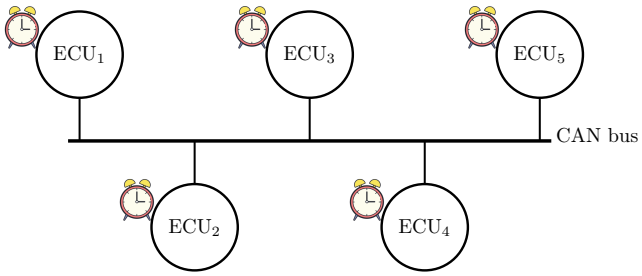


Figure 1: An example of CAN bus

approximate but faster version. The implementation of these analyses uses several undocumented optimizations.

In this section, we first present the notions of the CAN protocol which are useful for describing a CAN system. We refer interested readers to the official CAN specification [9] for more details; Then, we present the system model considered in CAN analyses as well as notations and definitions used throughout this article.

2.1 CAN protocol

The CAN protocol describes communication rules between ECUs on a CAN bus. The topology of a CAN bus is illustrated in Figure. 1. Each ECU uses a local clock and sends messages to other ECUs according to the FPNP scheduling policy. Each message is assigned a unique priority and cannot be preempted once its transmission has started. When several messages must be transmitted, the bus selects the message with the highest priority. Messages to be sent are encapsulated in a fixed frame. The frame consists of:

- a unique identifier (11 bits for the standard format and 29 bits for the extended format);
- a message to transfer whose size, noted m, can range from 1 to 8 bytes;
- some control bits *e.g.,* acknowledgment, error detection, bit stuffing, *etc.*

In the worst case, the size of a frame is $55 + 10m$ bits for the standard format and $80 + 10m$ bits for the extended format [11].

Maximum transmission time

Usually, the bit rate of a CAN bus is fixed and can be either 125 kbit/s, 250 kbit/s, 500 kbit/s, or 1 Mbit/s. With the above information, we can determine the maximum transmission time of a message. For instance, on a 500 kbit/s CAN bus, the transfer of an 8-byte message using the standard frame format takes at most

$$\frac{(55 + 10 * 8)\,\text{bits}}{500\,\text{kbit/s}} = 270\ \mu\text{s}$$

This formula is used later to produce task sets for experimental evaluations.

Response time

In many cases, a message cannot be immediately transmitted after its activation (*i.e.,* its request to send). The bus may already be busy sending another message, and then all higher priority messages that are also activated will be transmitted first. The response time of a message is defined as the time duration between its activation and its completion (*i.e.,* its end of transmission). It is usually associated with a real-time constraint stating that its response time should be less than a given duration (*i.e.,* its relative deadline).

Fixed timing relation between messages within one ECU

Each ECU contains several periodic messages to send. All activation times of a periodic message are known once its first activation time is fixed. When all messages activate at the same time, the message with the lowest priority can be delayed for a considerable duration. Consequently, it could result in missing its deadline. To address this issue, messages activations are separated by introducing an *offset* for each message. It is a fixed time duration from local time 0 to the first activation time of a message. Adding offsets allows to shift periods, to increase the bus utilization while keeping the system schedulable.

2.2 System model

The system model considered consists of a set of transactions (representing ECU nodes)

$$Sys := \{T_1, T_2, \ldots, T_N\}$$

where each transaction T_i is a set of periodic tasks (representing messages):

$$T_i := \{\tau_{i,1}, \tau_{i,2}, \ldots, \tau_{i,M}\}$$

Each task $\tau_{i,k}$ has a fixed and unique priority k (a smaller number means a higher priority) and is characterized by a 4-tuple

$$(c^+_{i,k}, d_{i,k}, p_{i,k}, o_{i,k})$$

where

- $c^+_{i,k}$ denotes its worst-case execution time (WCET), *i.e.,* the maximum transmission time,
- $d_{i,k}$ its relative deadline,
- $p_{i,k}$ its activation period, and
- $o_{i,k}$ its *offset*, *i.e.,* the duration between an activation of T_i and the first activation of $\tau_{i,k}$ after that. In this paper, we consider only *constrained* offsets, *i.e.,* $o_{i,k} < p_{i,k}$.

Each transaction T_i is activated periodically after an offset o_i with a period (called the hyper-period) equal to the least common multiple of the periods of all its tasks. Formally,

$$HP_i = lcm\{p_{i,1}, p_{i,2}, \ldots, p_{i,M}\}$$

Tasks within the same transaction share the same clock. All tasks of T_i being periodic, their offsets define a precise timing relation between them. Note that the model does not suppose any global synchronization between transactions. Any possible time shift between any two transactions is assumed to be possible and must be considered by the analysis. Task $\tau_{i,k}$ activates periodically its jobs at $o_{i,k} + m \times p_{i,k}$ with $m \geq 0$. A job $\jmath$ of a task $\tau_{i,k}$ is characterized by

- its activation time $act(\jmath)$,
- its completion (end of transmission) time $end(\jmath)$ and
- its execution (transmission) time $c(\jmath)$ ($\leq c^+_{i,k}$)

Its response time $r(\jmath)$ is defined as $end(\jmath) - act(\jmath)$. The worst-case response time (WCRT) of task $\tau_{i,k}$, denoted $wcrt_{i,k}$, is the largest possible response time among all jobs of task $\tau_{i,k}$.

In the following, we mostly use terminology related to tasks rather than messages, since our analysis applies to any FPNP scheduling of tasks with transactions.

We note $hep(k)$, $hp(k)$, $lp(k)$ the sets of tasks of the system under study whose priorities are higher than or equal to, higher than or lower than k, respectively.

3 Certified RTAs for CAN

In this section, we describe two RTAs for CAN, a precise analysis and an approximate one, that we use to certify the results of the RTaW-Pegase tool. The correctness of these RTAs has been proven using the Coq proof assistant [4] on top of the Prosa library [1]. In the following, we consider a task $\tau_{i,k}$ and describe how the two RTAs compute an upper bound on its worst-case response time. The presentation follows the Coq specification. We omit proofs of lemmas and theorems and refer the interested reader to the Coq source [2].

Both analyses operate on a set of scenarios. A scenario represents an alignment between among transactions that corresponds to a set of maximum workload functions, which is used to compute an upper bound on the response time of $\tau_{i,k}$ for that alignment. The precise analysis considers the set of precise scenarios corresponding to all possible alignments between transactions. The approximate analysis considers a much smaller set of approximate scenarios where all transactions, except the one of the task under study, are represented by an approximate workload function.

Both analyses rely heavily on the well-known concept of busy window, as we explain now.

3.1 Busy window analysis

Let us consider a task $\tau_{i,k}$ for which we want to bound the response time.

Definition 1 (Level-k quiet time)**.** *An instant t is said to be a* level-k quiet time *if all jobs of priority higher than or equal to k activated strictly before t have completed at t.*

Definition 2 (Level-k busy window)**.** *A time interval $[t_1, t_2[$ is said to be a* level-k busy window *if:*

1. *t_1 and t_2 are level-k quiet times;*
2. *there is no level-k quiet time in $]t_1, t_2[$; and*
3. *at least one job with a priority higher than or equal to k is activated in*[3] *$[t_1, t_2[$.*

Note that, due to the non-preemptive nature of scheduling, a job with priority lower than k may be executing at the beginning of a level-k busy window. This is referred to as *blocking* and the duration of the blocking delay at the beginning of a busy window $[t_1, t_2[$ is denoted $b_k(t_1)$.

[3]Using this definition, we can prove that at least one job with a priority higher than or equal to k is activated at t_1.

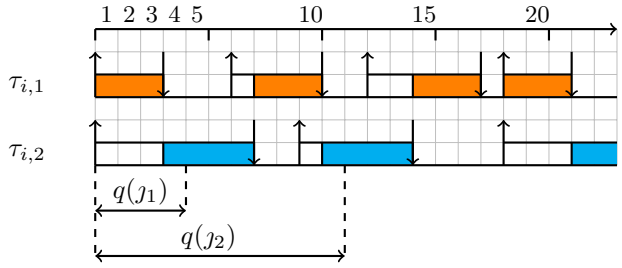


Figure 2: Queueing delays in a busy window.

Let $\jmath$ be an arbitrary job of task $\tau_{i,k}$. Let $[t_1, t_2[$ denote the level-k busy window during which $\jmath$ of $\tau_{i,k}$ is activated[4]. Clearly, a job with a priority higher than or equal to k completes in the level-k busy window in which it is activated. In other words, its response time can be bounded by the length of the corresponding busy window. Such a bound is however quite coarse. In particular, there may be several jobs of the same task activated in the same busy window. To provide a tighter bound on the response time of $\jmath$, we introduce the notions of phase and queueing delay, similar to their definition in [15].

Definition 3 (Queueing delay)**.** *The* queueing delay *of a job $\jmath$ in a level-k busy window $[t_1, t_2[$, denoted $q(\jmath)$, is the duration $t_\jmath - t_1 + 1$ where $t_\jmath$ is the instant at which $\jmath$ is scheduled for the first time (*i.e., *starts execution).*

Figure 2 shows the queueing delays of the two jobs of task $\tau_{i,2}$ in a level-2 busy window where $\jmath_n$ denotes the n-th job of task $\tau_{i,2}$ activated in that busy window.

Definition 4 (Phase)**.** *The* phase *of a job $\jmath$ activated in an interval $[t_1, t_1 + \Delta[$, denoted $\varphi(\jmath)$, is the duration $act(\jmath) - t_1$,* i.e., *the duration between t_1 and the activation time of that job. The* phase *of a task $\tau_{j,l}$ is the phase of the first job of $\tau_{j,l}$ in $[t_1, t_1 + \Delta[$.*

For instance, in Figure 2, the phase of $\jmath_1$ in [0,23[is 0, and that of $\jmath_2$ is $0 + 1 \times 9 = 9$.

Based on these definitions, $\jmath$ completes at

$$t_1 + q(\jmath) - 1 + c(\jmath)$$

The response time of $\jmath$ is therefore

$$r(\jmath) = t_1 + q(\jmath) - 1 + c(\jmath) - act(\jmath)$$

which can be rewritten with the phase of $\jmath$

$$r(\jmath) = q(\jmath) - 1 + c(\jmath) - \varphi(\jmath) \tag{1}$$

From now on, our objective will be to compute a bound on $r(\jmath)$. For that, we will look for an upper bound on $q(\jmath)$ and $c(\jmath)$ as well as a lower bound on $\varphi(\jmath)$. First, let us explain how the queueuing delay $q(\jmath)$ can be computed, using the notion of workload.

Definition 5 (Workload)**.** *The* workload *$wl_{j,l}(t_1, \Delta)$ of task $\tau_{j,l}$ in a time interval $[t_1, t_1 + \Delta[$ is the cumulative execution time of its jobs activated in that interval.*

[4] It can be shown that if the utilization is below 100%, it is always possible to compute that level-k busy window.

The queueing delay of $\jmath$ can be found by computing the least fixed point of the following equation:

$$\Delta = f_q(\Delta)$$

where

$$f_q(\Delta) := b_k(t_1) + \sum_{\tau_{j,l} \in hp(k)} wl_{j,l}(t_1, \Delta) + wl_{i,k}\left(t_1, \varphi(\jmath)\right) + 1 \tag{2}$$

The intuition behind the above formula is that job $\jmath$ can only be scheduled after (1) the initial blocking delay; (2) the execution of all higher priority jobs that have been activated before $\jmath$ can get scheduled; and (3) the execution of other jobs of $\tau_{i,k}$ activated before $\jmath$.

The key to upper bound $q(\jmath)$ is to upper bound $f_q(\Delta)$ by a function $f_q^+(\Delta)$, and then use the following lemma to conclude that the fixed point of $f_q^+(\Delta)$ is an upper bound on $q(\jmath)$.

Lemma 1. *Let $f, g : \mathbb{N} \to \mathbb{N}$ be two monotonically increasing functions and Δ_1 and Δ_2 be fixed points of the equations $\Delta = f(\Delta)$ and $\Delta = g(\Delta)$, respectively. Then if for all $x : \mathbb{N},\ f(x) \leq g(x)$ and, for all $x : \mathbb{N}^+,\ x < \Delta_1$, we have $x < f(x)$, then $\Delta_1 \leq \Delta_2$.*

To upper bound $f_q(\Delta)$, let us start by upper bounding the blocking delay.

Lemma 2 (Blocking delay bound)**.** *$b_k(t_1) \leq b_k^+$ where*

$$b_k^+ := \max_{\tau_{j,l} \in lp(k)} (c_{j,l}^+ - 1) \tag{3}$$

Indeed, blocking is maximized when the lower priority task that has the largest worst-case execution time starts executing a job with such an execution time just one time unit before the start of the level-k busy window.

Second, let us upper bound the workload of tasks. For any task $\tau_{j,l}$ in the system, and for any time duration Δ, the workload of task $\tau_{j,l}$ is maximized when all its jobs take their WCET. Formally,

Lemma 3 (Workload bound)**.** *$wl_{\tau_{j,l}}(t_1, \Delta) \leq wl_{\tau_{j,l}}^+(t_1, \Delta)$ where*

$$wl_{\tau_{j,l}}^+(t_1, \Delta) := \lceil \frac{\Delta - \varphi(\tau_{j,l})}{p_{j,l}} \rceil \times c_{j,l}^+ \tag{4}$$

$wl_{\tau_{j,l}}^+$ is called the workload bound function of task $\tau_{j,l}$.

Let us now focus on lower bounds on the phase. Suppose that $\jmath$ is the n-th job of task $\tau_{i,k}$ in the interval $[t_1, t_1 + \Delta[$. Given that we only consider periodic tasks, we have

$$\varphi(\jmath) = \varphi(\tau_{i,k}) + (n-1) \times p_{i,k} \tag{5}$$

Within busy window $[t_1, t_2[$, the response time of task $\tau_{i,k}$ can be locally bounded by

$$\max_{n \leq n_\jmath^+} \{q(\jmath_n) - 1 + c(\jmath_n) - \varphi(\jmath_n)\}$$

where $n_\jmath^+$ is an upper bound on the number of jobs of task $\tau_{i,k}$ in that busy window and $\jmath_n$ represents the n-th job of task $\tau_{i,k}$ activated in that busy window. $n_\jmath^+$ can be computed by

$$n_\jmath^+ := \left\lceil \frac{bw_\jmath}{p_{i,k}} \right\rceil \tag{6}$$

where $bw_\jmath = t_2 - t_1$ denotes the duration of the busy window during which $\jmath$ executes.

$bw_\jmath$ can be found by computing the least fixed point of the following equation:

$$\Delta = f_{bw}(\Delta)$$

where

$$f_{bw}(\Delta) := b_k(t_1) + \sum_{\tau_{j,l} \in hep(k)} wl_{j,l}(t_1, \Delta)$$

Computing an upper bound on the WCRT of task $\tau_{i,k}$ amounts now to finding a finite set of scenarios such that $f_{bw}(\Delta)$ and $f_q(\Delta)$ for any busy window are bounded by the corresponding functions $f^+_{bw}(\Delta)$ and $f^+_q(\Delta)$ of a scenario in that set. The WCRT of the task $\tau_{i,k}$ is found by taking the maximum WCRT found for all these scenarios.

3.2 Precise analysis

The precise analysis considers the finite set of scenarios corresponding to the cases where

1. all jobs in the busy window take their worst-case execution time to complete; and
2. t_1 is aligned with an activation in each transaction.

We will show that, for any given job $\jmath$ of task $\tau_{i,k}$, there is a scenario belonging to the set described above that upper bounds $q(\jmath)$ and lower bounds $\varphi(\jmath)$.

Definition 6 (Alignment)**.** *Let t_j denote the time of the first activation in $T_j \cap hep(k)$ after t_1 and al_j, called alignment, the duration between the latest activation of transaction T_j before t_1, and t_j.*

For any task $\tau_{j,l}$ in the system, and for any time instant t_1 and time duration Δ, the workload of task $\tau_{j,l}$ is maximized when we right shift the interval $[t_1, t_1 + \Delta[$ to align t_1 with the first activation with a priority higher than or equal to k in T_j after t_1.

Lemma 4. $wl^+_{j,l}(t_1, \Delta) \leq wl^+_{j,l}(t_j, \Delta)$

The key to the precise analysis is that there is only a finite number of t_j for which $wl^+_{j,l}(t_j, \Delta)$ differs: at the end of the hyperperiod of T_j, the activation pattern, and therefore the workload, repeats. In fact, we can show that $wl^+_{j,l}(t_j, \Delta) = wl^+_{j,l}(o_j + al_j, \Delta)$ and that al_j is an element of Al_j defined as follows.

$$Al_j := \bigcup_{\tau_{j,l} \in T_j \cap hep(k)} \{\, al_j = o_{j,l} + m \times p_{j,l} \mid m \in \mathbb{N} \wedge al_j < HP_j \,\}$$

The list Al_j of offsets for each transaction T_j is composed of all possible activations of jobs with a higher priority than k ($\in hep(k)$) in the transaction T_j within its hyper-period HP_j.

The consequence of that is that the workload (and therefore the busy window length and the queueing delay) in any busy window can be upper bounded by a finite set of workload bounds.

Lemma 5. *For any time duration Δ,*

$$f_{bw}(\Delta) \leq f^+_{bw}(\Delta) \tag{7}$$

where

$$f^+_{bw}(\Delta) := b^+_k + \sum_{\tau_{j,l} \in hep(k)} wl^+_{j,l}(o_j + al_j, \Delta) \tag{8}$$

Lemma 6. *In any level-k busy window and for any time duration* Δ

$$f_q(\Delta) \leq f_q^+(\Delta) \tag{9}$$

where

$$f_q^+(\Delta) := b_k^+ + \sum_{\tau_{j,l} \in hp(k)} wl_{j,l}^+(o_j + al_j, \Delta) + wl_{i,k}^+(o_i + al_i, \varphi(\jmath)) + 1 \tag{10}$$

Furthermore, one can easily express $\varphi(\jmath)$ as a function of al_i. If $\jmath$ is the n-th job of $\tau_{i,k}$, then:

$$\varphi(\jmath) = o_{j,l} + (n-1) \times p_{i,k} - al_i \tag{11}$$

To upper bound the WCRT $wcrt_{i,k}$ of $\tau_{i,k}$, we thus need to test all possible such al_j for each transaction T_j, representing all possible alignments of the busy window with an activation. The list of all scenarios is made of all combinations of alignments over all transactions. The WCRT $wcrt_{i,k}$ of task $\tau_{i,k}$ is bounded by the maximal WCRT of all scenarios.

Definition 7 (Scenario)**.** *A scenario s is a tuple* $(al_1, \cdots, al_N)$ *representing a list of alignments for each transition. We denote* $r_s^+(\tau_{i,k})$ *the response time bound obtained using the corresponding bounds described above.*

Theorem 1. *Let* $\times$ *denote the cartesian product. Then*

$$wcrt_{i,k} \leq \max_{s \in \mathcal{S}} \{r_s^+(\tau_{i,k})\}$$

where $\mathcal{S} = Al_1 \times \cdots \times Al_N$

3.3 Approximate analysis

For large systems, the number of precise scenarios explodes and the precise analysis quickly becomes intractable. In this subsection, we present a more efficient but approximate analysis. It follows the same approach as presented in [24]. Its principle is to use approximate scenarios, which consist in the alignments of the considered transaction T_i only; the other transactions are represented by an approximate workload bound function. The latter provides an upper bound on the workload among all possible alignments and is defined as follows.

Definition 8. *The approximate workload bound function of a transaction* T_j *for the duration* Δ *is defined as the maximum workload among all possible alignments represented by* Al_j*:*

$$wl_{T_j}^*(\Delta) = \max_{al_j \in Al_j} \left\{ \sum_{\tau_{j,l} \in T_j \cap hep(k)} wl_{\tau_{j,l}}^+(o_j + al_j, \Delta) \right\}$$

Functions $f_{bw}^+(\Delta)$ and $f_q^+(\Delta)$ are upper bounded by using $wl_{T_j}^*(\Delta)$ for each transaction T_j. However, in order to obtain a tighter bound, we compute the precise workload of transaction T_i of task $\tau_{i,k}$ (*i.e.,* the task we analyze).

Lemma 7 (Bound-$f_{bw}^+(\Delta)$)**.** *For any time duration* Δ *and any* $al_i \in Al_i$

$$f_{bw}^+(\Delta) \leq f_{bw}^*(\Delta)$$

where

$$f_{bw}^*(\Delta) := b_k^+ + \sum_{\substack{T_j \in Sys \\ j \neq i}} wl_{T_j}^*(\Delta) + \sum_{\tau_{i,l} \in T_i \cap hep(k)} wl_{\tau_{i,l}}^+(o_j + al_j, \Delta)$$

Lemma 8 (Bound-$f_q^+(\Delta)$)**.** *For any time duration* Δ *and any* $al_i \in Al_i$

$$f_q^+(\Delta) \leq f_q^*(\Delta)$$

where

$$f_q^*(\Delta) := b_k^+ + \sum_{\substack{T_j \in Sys \\ j \neq i}} wl_{T_j}^*(\Delta) + \sum_{\tau_{i,h} \in T_i \cap hp(k)} wl_{\tau_{i,h}}^+(o_j + al_j, \Delta) + wl_{\tau_{i,k}}^+(o_i + al_i, \varphi(j)) + 1 \quad (12)$$

We compute $bw_{al_i}^*$ the least fixed point of equation

$$bw_{al_i}^* = f_{bw}^*(bw_{al_i}^*)$$

and, for each $n \leq n_{al_i}^*$, the least fixed point of equation:

$$q_{al_i,n}^* = f_q^*(q_{al_i,n}^*)$$

then, the response time of jobs of task $\tau_{i,k}$ activated in the busy window $[t_1, t_2[$ is upper bounded by $r_{al_i}^*(\tau_{i,k})$ defined as:

$$\max_{n \leq n_{al_i}^*} \left\{ q_{al_i,n}^* - 1 + c_{i,k}^+ - (o_{j,l} + (n-1) \times p_{i,k} - al_i) \right\}$$

Then, the WCRT $wcrt_{i,k}$ of task $\tau_{i,k}$ is the maximum of these values for all possible alignments represented by Al_i.

Theorem 2.

$$wcrt_{i,k} \leq \max_{al_i \in Al_i} \{r_{al_i}^*(\tau_{i,k})\}$$

Compared to the precise analysis, we do not consider all possible combinations (the cartesian product) of all alignments of all transactions.

3.4 Generic Analysis

We have presented two analyses for CAN:

- A precise analysis which provides a precise result but quickly becomes intractable.
- An approximate analysis which is able to efficiently return approximate results.

To benefit from the advantages of both the precise analysis and the approximate analysis, we combine the two analyses together in order to use as much as possible the approximate version to compute the precise results. The combined analysis presented in Section 4 relies on a generic analysis that we present now.

The generic analysis combines in a generic way the precise and approximate analyses. It allows computing the WCRT by using a combination of precise alignments and approximate alignments[5]. The precision of its results depends on its inputs. It computes the precise result if its inputs are all precise alignments, the approximate results while its inputs are all approximate alignments, or a trade-off between precise and approximate analysis if its inputs contain both precise and approximate alignments. It separately computes precise workloads and approximate workloads. Transactions in systems are divided into two disjoint sets:

[5] An approximate alignment means that an approximate workload bound function are used for computing the WCRT

1. SET_p (Contributing precise workload); Each transaction in SET_p provides a precise workload for each specific alignment between transactions among this set. All possible alignments are considered to perform the worst-case response time for a given task $\tau_{i,k}$ to be analyzed. Note that the transaction T_i containing the task under consideration $\tau_{i,k}$ will always be in this set.
2. SET_a (Contributing approximate workload). Each transaction in SET_a provides an approximate workload as defined in Definition 8.

Considering the two sets, functions $f^+_{bw}(\Delta)$ and $f^+_q(\Delta)$ can be upper bounded by using $wl^+(\Delta)$ for transactions from SET_p and $wl^*(\Delta)$ for transactions from SET_a.

Definition 9. *Let $\mathcal{S}_{\text{SET}_p} := \{\cdots \times Al_j \times \cdots\}$ be all combinations of alignments among transactions in SET_p, where Al_j is the set of alignments for transaction $T_j \in \text{SET}_p$.*

Lemma 9 (Refined Bound-$f^\star_{bw}(\Delta)$)**.** *For any time duration Δ and any $s \in \mathcal{S}_{\text{SET}_p}$:*

$$f^+_{bw}(\Delta) \leq f^\star_{bw}(\Delta)$$

where

$$f^\star_{bw}(\Delta) := b^+_k + \sum_{T_i \in \text{SET}_a} wl^*_{T_i}(\Delta) + \sum_{\substack{T_j \in \text{SET}_p \\ o \in O}} wl^+_{T_j}(o, \Delta)$$

Lemma 10 (Refined Bound-$f^\star_q(\Delta)$)**.** *For any time duration Δ, any $s \in \mathcal{S}_{\text{SET}_p}$ and n:*

$$f^+_q(\Delta) \leq f^\star_q(\Delta)$$

where

$$\begin{aligned} f^\star_q(\Delta) := b^+_k &+ \sum_{T_a \in \text{SET}_a} wl^*_{T_a}(\Delta) + \sum_{\substack{T_p \in \text{SET}_p \\ p \neq i}} wl^+_{T_p}(o_p + al_p, \Delta) \\ &+ \sum_{\tau_{i,l} \in T_i \cap hep(k)} wl^+_{\tau_{i,h}}(o_i + al_i, \Delta) + wl^+_{\tau_{i,k}}(o_i + al_i, \varphi(\jmath_n)) + 1 \end{aligned} \tag{13}$$

and $\varphi(\jmath_n) := o_{j,l} + (n-1) \times p_{i,k} - al_i$.

Let Al be a list of alignments which consists of one al_j for each transaction $T_j \in \text{SET}_p$. We can prove that $Al \in \pounds_{\text{SET}_p}$. Similar to the two previous analyses, an upper bound of the length of $bw_\jmath$ is obtained by computing $bw^\star_{Al}$, the least fixed point of equation

$$bw^\star_{Al} = f^\star_{bw}(bw^\star_{Al})$$

Within $bw^\star_{Al}$, we can compute the maximum number $n^\star_{Al}$ of activations of task $\tau_{i,k}$. Then, for each $q \leq n^\star_{Al}$, we compute the least fixed point of equation:

$$q^\star_{Al,n} = f^\star_q(q^\star_{Al,n})$$

Consequently, the response time of jobs of task $\tau_{i,k}$ activated in the busy window $[t_1, t_2[$ is upper bounded by

$$r^\star_{Al}(\tau_{i,k}) = \max_{n \leq n^\star_{Al}} \left\{ q^\star_{Al,n} - 1 + c^+_{i,k} - \varphi(\jmath_n) \right\}$$

with $\varphi(\jmath_n) := o_{j,l} + (n-1) \times p_{i,k} - al_i$.

Finally, the WCRT $wcrt_{i,k}$ of task $\tau_{i,k}$ is the maximum of these values for all possible alignments represented by $\mathcal{S}_{\text{SET}_p}$.

Theorem 3.

$$wcrt_{i,k} \leq \max_{s \in \mathcal{S}_{\text{SET}_p}} r_s^\star(\tau_{i,k})$$

This analysis is a generic version for both the precise one and the approximate one. It provides with the same result as the precise analysis when SET_p only contains transaction T_i, while it returns the same result as the approximate analysis when $\text{SET}_a = \emptyset$. Of course, many other combinations can be used by using different divisions of transactions.

The complexity of any instance of the generic analysis lies between the complexities of the precise and approximate analyses. In Theorem 3, the size of $\pounds_{\text{SET}_p}$ reflects the time complexity of this analysis. It is greater than the size of Al_i used for the approximate analysis and less than the size of $\{Al_1 \times \cdots \times Al_N\}$ used for the precise analysis. It makes it possible to obtain many different precisions by choosing how to divide transactions into SET_a and SET_p.

4 Combined RTA and Result Certifier

In this section, we present two combined RTAs based on the previous generic analysis. The underlying idea is to use the approximate version when it can be shown that its result is the precise one. In addition, we present CertiCAN, a result certifier based on combined RTA, which can certify the results of industry analyzers even for large systems.

For the sake of clarity, we start by presenting a simple two-level combined RTA. Then, we present the full combined RTA which is at the basis of CertiCAN.

4.1 Two-level Combined RTA

The two-level combined analysis is based on the precise and approximate analyses presented in Sections 3.2 and 3.3. Its main features are the following.

- It uses the approximate analysis as much as possible to avoid unnecessary computations and thus increase performance;
- It nevertheless computes the same results as the precise analysis.

For the sake of simplicity, we abstract an analysis as a function $\mathcal{A}$ that takes a scenario s as input and that returns a result *i.e.,* the WCRT. The precise analysis, $\mathcal{A}_p$, considers a list of precise scenarios S_p and computes a worst case response time for each. Its result is the maximum of all these computations, that is:

$$\max_{s_p \in S_p} \mathcal{A}_p(s_p)$$

The approximate analysis, $\mathcal{A}_a$, does the same on a list of approximate scenarios S_a. The two soundness properties of the approximate analysis are:

- Each approximate scenario s_a *dominates* a list of precise scenarios written $Dom(s_a)$ *i.e.,*

$$max_{s_p \in Dom(s_a)} \mathcal{A}_p(s_p) \leq \mathcal{A}_a(s_a) \tag{14}$$

- The list S_a of approximate scenarios dominates all precise scenarios of S_p.

Therefore, we know that the result of the approximate analysis is an over approximation of the precise result, that is

$$max_{s_p \in S_p} \mathcal{A}_p(s_p) \leq max_{s_a \in S_a} \mathcal{A}_a(s_a) \tag{15}$$

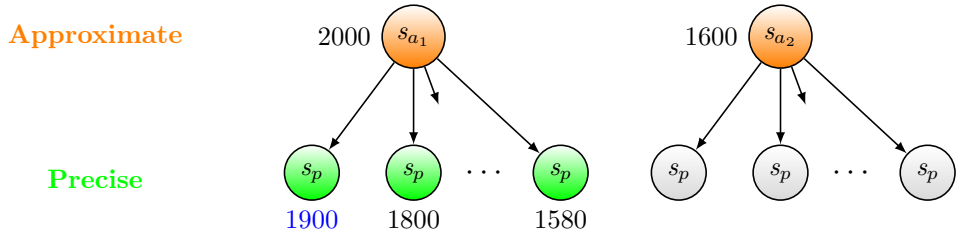


Figure 3: Scenario domination according to the two-level combined analysis for a system of four transactions. Each approximate scenario (orange node) dominates 1000 precise scenarios (green nodes). Unprinted nodes and the nodes in gray represent the scenarios which do not need to be analyzed.

The two-level combined analysis is based on the following observation: If the WCRT obtained for an approximate scenario s_a is less than the WCRT found so far on the set of precise scenarios visited, then there is no need to analyze the precise scenarios dominated by s_a.

Example 1. *Consider a system of four transactions* $\{T_1, T_2, T_3, T_4\}$ *and the number of alignments in each transaction is 2, 10, 10, 10 respectively. For a task of transaction* T_1*, the precise analysis has to perform 2000 scenarios for computing a precise result, while we only need to examine 2 scenarios for obtaining an approximate result by over-approximating the three other transactions* T_2*,* T_3*, and* T_4*.*

This example with 2 approximate scenarios and each one dominating 1000 precise scenarios is depicted in Figure 3, where vertices represent scenarios, edges represent the domination relation and labels next to vertices are their corresponding response times. For this example, the two-level combined analysis is called with an initial WCRT 0 *and the list* $[(2000, s_{a_1}); (1600, s_{a_2})]$*. It proceeds as follows:*

- *the current approximate response time (*2000*) is greater than the current WCRT (*0*) so the maximum response time of the* 1000 *dominated precise scenarios is computed (*1900*) and becomes the current WCRT;*
- *because* 1900 *is greater than or equal to the next approximate response time in the list (*1600*), the response times of the corresponding dominated scenarios do not need to be computed (they are necessarily smaller);*
- *the analysis returns* 1900 *which is the precise WCRT but considered only* 1000 *precise scenarios instead of* 2000*.*

The two-level combined RTA returns the same WCRT as the precise analysis but using in practice much fewer computations.

4.2 Full Combined RTA

The problem of the two-level combined RTA is that a single approximate scenario may dominate a considerable number of precise scenarios, in particular, for systems that have many transactions. Therefore, for large systems, it is sometimes intractable to analyse all the precise scenarios dominated by the first approximate scenario alone. In order to solve this issue, we add intermediate approximate levels to get a multi-level combined RTA. We refer to this version as the full combined RTA.

The full combined analysis is based on the generic analysis as presented in Section 3.4. It separates the transactions into two disjoint sets SET_p and SET_a. It examines all alignments for transactions in SET_p and uses the approximate workload bound function for each transaction in SET_a. The main idea of the full combined analysis is to refine the approximate analysis transaction by transaction, until the computed result cannot decrease any more, and is then guaranteed to be the same as that of the precise analysis.

For instance, consider a system $\{T_1, \ldots, T_n\}$; to analyze a task in transaction T_1, the full combined analysis starts with $\text{SET}_p = \{T_1\}$ and $\text{SET}_a = \{T_2, \ldots, T_n\}$. The scenarios correspond to all combinations $Al^{\star}_{\text{SET}_p}$ of alignments among transactions in SET_p, that is represented by Al_1; Then, we refine the result by putting one transaction from SET_a into SET_p, for example $\text{SET}_p = \{T_1, T_2\}$ and $\text{SET}_a = \{T_3, \ldots, T_n\}$. Thus, we must now check more scenarios $Al^{\star}_{\text{SET}_p} = Al_1 \times Al_2$; Consequently, the time complexity increases but the result is more precise. When all transactions are moved into SET_p, the analysis checks all combinations of alignments, that is, all precise scenarios. Therefore, it provides the same result as the precise analysis. Using this refinement technique, we build a tree with as many levels as the number of transactions in the system to analyze. Each level represents a setting of the generic analysis. The first level describes the approximate analysis, while the last level expresses the precise analysis.

Example 2. *We take the same example with four transactions as in Example 1 and build a 4-level tree according to the procedure of the full combined analysis (in Figure 4). The full combined analysis first computes the sorted list* $[(2000, s_{a_1}); (1600, s_{a_2})]$ *representing scenarios at Level-1 and the transaction sets* $\text{SET}_p := \{T_1\}$ *and* $\text{SET}_a := \{T_2, T_3, T_4\}$ *to refine. Given that* SET_a *is not empty, we move one transaction from* SET_a *to* SET_p *and compute the results for the 10 scenarios at Level 2 dominated by the scenario corresponding to the largest approximate response time (*2000*) at Level 1. Then the results for the 10 scenarios are sorted by descending order. That builds a part of Level-2; Recursively, we take one transaction from* SET_a *to compute Level-3, and so on until Level-4 when the* SET_a *is empty; At Level-4, all results are precise and the largest response time is* 1900*. Then,* 1900 *becomes the current precise WCRT; because* 1900 *is greater than or equal to all next approximate response at any level (*1890 *at Level 3,* 1700 *at Level 2,* 1600 *at Level 1), the response times of the corresponding dominated scenarios do not need to be computed (they are necessarily smaller); the analysis returns* 1900 *which is the precise WCRT. It had to consider only* 22 *approximate and* 10 *precise scenarios.*

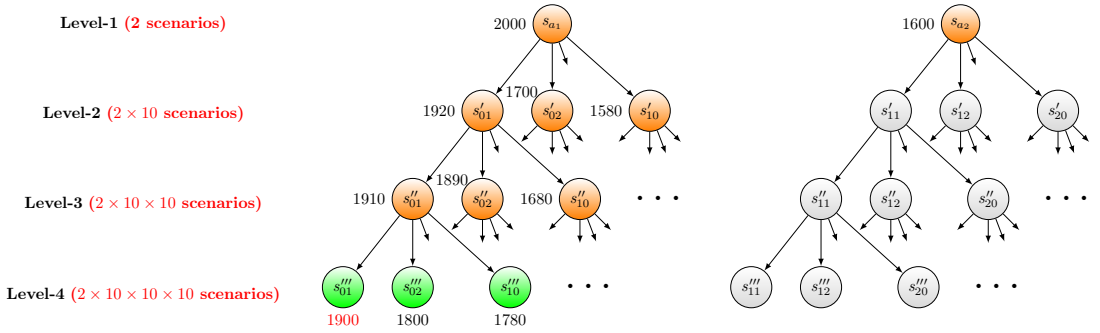


Figure 4: Scenario domination according to the full combined analysis for a system of four transactions. Each approximate scenario dominates 10 less approximate scenarios. Unprinted nodes and the nodes in gray represent the scenarios which do not need to be analyzed.

The procedure of the full combined analysis is similar to the two-level combined analysis: It uses the computed results from approximate scenarios to help the refinement procedure then to

reduce the number of scenarios to analyze. Generally, it can be seen as a branch-and-bound algorithm applied to a tree. The efficiency of the analysis depends on the order in which transactions are chosen for the refinement. We discuss this point in Section 5.3.

Algorithm 1 Full Combined RTA

```
1: % R denotes the current WCRT, its initial value is 0
2: % l_S denotes a list of scenarios (at the first level) paired with their corresponding WCRT
3: l_S := sort(map(λs.(A_g(s), s)) S_1)
4: % l_tr denotes the list of transactions to be refined (i.e., SET_a)
5: procedure CRTA(R, l_S, l_tr)
6:     match l_S with
7:     % l_S empty: returns the WCRT
8:     | nil ⇒ return R
9:     % otherwise: takes one element to analyze
10:    |(r, s) :: l'_S ⇒
11:    % if the current WCRT is greater than the approximate result r,
12:    then returns the current WCRT
13:    if R ≥ r then return R
14:    else
15:        match l_tr with
16:        % if l_tr is empty, returns max(r, R)
17:        | nil ⇒ return max(r, R)
18:        % otherwise, takes one transaction to refine
19:        |t :: l'_tr ⇒
20:        % computes the dominated scenarios as well as their results,
21:        and sorts their results
22:          l_local ← (sort (map (λs.(A_g(s), s)) (refine s l_tr)));
23:        % computes the result for those dominated scenarios
24:          R_local ← CRTA(R, l_local, l'_tr);
25:        % recursively, analyzes the remaining elements l'_S of the list l_S
26:          CRTA(R_local, l'_S, l_tr)
27:    end if
28: end procedure
29: CRTA(0, l_S, l_tr)
```

The structure of the full combined analysis is shown in Algorithm 1. First, the generic analysis is first applied to each scenario at the first level (we denote S_1 the set of scenarios at the first level). These results (*i.e.,* one WCRT for each scenario) paired with their corresponding scenario are sorted in descending order $l_S := sort\ (map\ (\lambda s.(\mathcal{A}_g(s), s))\ S_1)$ as shown at Line 3 in Algorithm 1. Sorting the list in that order leads to considering the scenario with the largest approximate WCRT first. This heuristic relies on the intuition that the largest precise WCRT (which is the value to be found) is more likely to be dominated by a large approximate WCRT and therefore, will be found earlier with that ordering.

Then, at Line 5 in Algorithm 1, the combined RTA (CRTA) is called with 0 as the initial result (noted R, it will be updated after each iteration), the list $l_S := sort\,(map\,(\lambda s.(\mathcal{A}_g(s), s))\ S_1)$, and a list l_{tr} of transactions to be refined (*i.e.,* $\textsc{Set}_a$). CRTA considers each approximate WCRT of list l_S in turn and starts with the first member (r, s) of l_S. Note that $r = \mathcal{A}_g(s)$ is the WCRT for scenario s. If $R \geq r$ then it stops and returns R (it is not the case when $R = 0$ as the initial input). Otherwise, it examines whether there are still some transactions in l_{tr} to be

refined (Line 15). If l_{tr} is empty, it means that all transactions have been taken into account precisely and the current WCRT becomes $max(r, R)$. If there are still transactions in l_{tr}, for the current scenario s, at Line 22, the function *refine* computes its dominated scenarios as well as their results using the refinement mechanism and sorts their results by descending order. These dominated scenarios paired with their results are stored in the list l_{local}. Next, the local WCRT, noted R_{local}, is computed for that list. Recursively, CRTA proceeds with the next element of the initial list and the local result R_{local} until it finds the global WCRT, *i.e.,* the precise WCRT.

We have formally proven in Coq that the results returned by the full combined analysis are the same as the ones computed by the precise analysis.

4.3 The CertiCAN Result Certifier

From the full combined RTA, we derive our result certifier, CertiCAN, which is able to check results of CAN analysis tools. Consider a result R_0 computed by an industrial analyzer. To certify this result, we apply the full combined analysis with an initial WCRT set at R_0.

The algorithm of CertiCAN is shown in Algorithm 2. To check that R_0 is equal to or larger than the precise WCRT, CertiCAN considers each approximate WCRT of the argument list (*i.e.,* l_S), in turn. If the current WCRT is equal to or less than R_0 then the certification is completed (and returns *True*) since all remaining approximate WCRTs (and the WCRTs of the corresponding dominated scenarios) of the list l_S are also less than R_0. Otherwise, it examines whether there are still some transactions in l_{tr} to be refined. If l_{tr} is empty, it means that there is no transaction left to be refined and the current WCRT is a precise result, consequently if R_0 is smaller than that precise WCRT, CertiCAN returns *False*. Otherwise, for the scenario s, the function *refine* computes the dominated scenarios as well as their corresponding WCRT. These dominated scenarios paired with their corresponding WCRT are stored in the list l_{local}. Then, CertiCAN checks results computed using that list. If the result is *False* then the certification procedure completes and returns *False*. Otherwise, CertiCAN proceeds, recursively, with the next element of the initial list l_S.

If CertiCAN returns *True* then it can be formally proven that R_0 is greater than or equal to the precise WCRT.

5 Optimization

Although the full combined RTA is able to accelerate the computation of a precise result, it is still time consuming when systems are complex. In this section, we present three additional optimizations for CertiCAN. They are based on the following observations:

1. Within one transaction, there are possibly many alignments. Each alignment within a transaction can be considered as a specific workload function to analyze as shown in Figure 5a. Therefore, we can determine the domination relation between alignments by comparing their corresponding workload functions. This permits to remove dominated alignments without loss of precision;

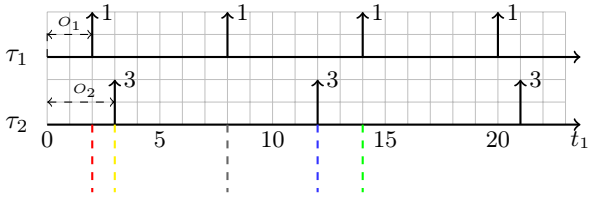
2. During the analysis, some functions are called with the same arguments repeatedly. These recomputations can be avoided using memoization and lookup tables;

3. The order of transactions considered for refinement affects the analysis speed. We have investigated several heuristics to find orders that accelerate the analysis.

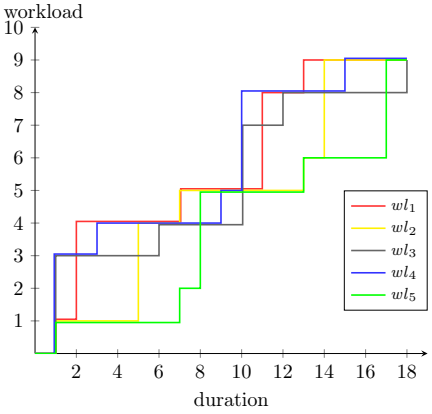
Algorithm 2 The CertiCAN Result Certifier

R_0 contains the WCRT to certify

```
1: % l_S denotes a list of scenarios (at the first level) paired with their corresponding WCRT
2: l_S := sort(map(λs.(A_g(s), s)) S_1)
3: % l_tr denotes a list of transactions (i.e., SET_a)
4: procedure CERTICAN(l_S, l_tr)
5:     match l_S with
6:     % if l_S is empty, returns True
7:     | nil ⇒ return True
8:     % otherwise, takes one element of l_S to analyze
9:     |(r, s) :: l'_S ⇒
10:    % if the WCRT to certify is greater than the approximate result r,
11:    then returns True
12:    if R_0 ≥ r then return True
13:    else
14:        match l_tr with
15:        % if l_tr is empty, returns False
16:        | nil ⇒ False
17:        % otherwise, takes one transaction to refine
18:        |t :: l'_tr ⇒
19:        % computes the dominated scenarios as well as their results,
20:        and sorts their results
21:          l_local ← (sort (map (λs.(A_g(s), s)) (refine s l_tr)));
22:        % certifies the result for those dominated scenarios
23:          R_local ← CERTICAN(l_local, l'_tr);
24:        if R_local = False then return False
25:        else
26:            % recursively, certifies the results for the remaining elements l'_S
27:            CERTICAN(l'_S, l_tr)
28:        end if
29:    end if
30: end procedure
31: CertiCAN((sort (map (λs.(A_g(s), s)) S_1)), l_tr)
```



(a) A transaction of two periodic tasks. Their offsets are 2 and 3, their periods are 6 and 9, respectively. Numbers next to arrows represent job execution times. The LCM of tasks periods is 18, during which there will be 5 activations. This transaction, therefore, represents 5 alignments. Each dashed line shows the beginning of an alignment.



(b) Workload functions of the five alignments of the transaction presented in Figure 5a.

Figure 5: Abstraction of one transaction to workload functions.

5.1 Removing dominated alignments

Each alignment of a transaction corresponds to a workload function as shown in Figure 5. For instance, for the alignment represented by the red dashed line (at time 2) in Figure 5a, its cumulated workload for a given duration Δ is $wl_1(\Delta)$ in Figure 5b. Thus, a transaction can be considered as a set of workload functions to analyze. We design an algorithm that filters out dominated alignments (*i.e.,* workload functions) from each transaction. For example, we can remove function wl_5 because its value is always smaller than that of wl_1. Then we prove that this filtering is correct, that is, it preserves the final results.

First, let us introduce some definitions expressing relations between workload functions and between scenarios.

Definition 10 (Strong workload domination)**.** *A workload function wl_1 is said to strongly dominate the workload function wl_2, denoted $wl_1 \succeq wl_2$, if and only if for any duration $\Delta \in N$,*

$$wl_1(\Delta) \geq wl_2(\Delta)$$

Definition 11 (Weak workload domination)**.** *A workload function wl_1 is said to weakly dominate the workload function wl_2* w.r.t. *a duration L, denoted $wl_1 \underset{L}{\succeq} wl_2$, if and only if for any duration $\Delta \in [0, L]$,*

$$wl_1(\Delta) \geq wl_2(\Delta)$$

Note that to determine the weak domination relation between workload functions for a set of periodic tasks, it is sufficient to compare their workload value for any duration within the hyper-period, *i.e.,* LCM of all task periods.

A scenario corresponding to a specific alignment can be considered as a collection of workload functions, which consists of one function from each transaction.

We now introduce the notion of scenario workload and generalize the notion of domination between scenarios.

Definition 12 (Scenario workload)**.** *Consider a scenario s corresponding to a collection of n workload functions $\{wl_1, wl_2, \ldots, wl_n\}$, for a given time duration Δ, its workload is the sum of*

all workloads provided by the n workload functions. Formally,

$$wl^s(\Delta) = \sum_{i=1}^{n} wl_i(\Delta)$$

Definition 13 (Scenario domination)**.** *A scenario s_1 is said to dominate the scenario s_2, denoted $s_1 \succeq s_2$, if and only if the WCRT computed for s_1 is greater than the one obtained for s_2. Formally,*

$$s_1 \succeq s_2 \iff WCRT_{s_1} \geq WCRT_{s_2}$$

Lemma 11 (Workload function domination implies scenario domination)**.** *Consider two scenarios s_1 and s_2 with workload functions wl^{s_1} and wl^{s_2}, respectively. Let L be the least fixed point of equation $f(L) = b_k^+ + \sum_{Tr \in Sys} wl_{Tr}^*(L)$. If wl^{s_1} weakly dominates wl^{s_2}* w.r.t. *L, then $\tilde{RT}_{s_1}^+ \geq \tilde{RT}_{s_2}^+$. As a result,*

$$wl^{s_1} \underset{L}{\succeq} wl^{s_2} \implies s_1 \succeq s_2 \tag{16}$$

In order to determine the domination relation between scenarios, we compute the weak domination relation between their corresponding workload functions. Then, we design a procedure to filter out dominated functions using domination relations and prove its correctness.

That algorithm relies on three simple functions:

1. *Filter*(l_f, f) removes the function f from the list l_f of functions if f is a member of that list;
2. *Compare*(wl_1, wl_2, L) computes whether wl_1 is weakly dominated by wl_2 *w.r.t.* L. It returns *true* if wl_1 is dominated by wl_2 for any $\Delta \in [0, L]$ and *false* otherwise;
3. *DominatedByOthers*(f, l_f, L), defined in Algorithm 3, computes whether a function f is weakly dominated (*w.r.t.* L) by any function except itself from a list l_f of functions.

The main procedure, described in Algorithm 4, proceeds as follows:

- It starts with n workload functions $f_0, .., f_{n-1}$ to be filtered and a length L for computing their weak domination relations;
- DominantFunction is called to filter out all dominated functions from l_f (Line 15). It takes two lists (initially, they are the same *i.e.,* l_f) and a number L; it returns the list of all dominant functions of l_f (*i.e.,* the functions which are not dominated by any other function from l_f);
- if l_2 is empty then it returns the list l_1 (Line 4), otherwise it checks whether the first function wl from l_2 is dominated by any other function from l_1 (Line 8);
- if wl is dominated by another function of l_1 then it removes wl from l_1 and continues to examine the functions l_2' by recursively calling DominantFunction (Line 9); otherwise,
- otherwise, it examines the remaining functions l_2' without changing l_1 (Line 11).

As mentioned before, the LCM of task periods is one sufficient length for determining weak domination relation among workload functions. In order to increase the efficiency of our tool, we have proven the smallest sufficient length (SSL)[6] for filtering out the dominated workload functions is the worst-case busy window. We will compare the efficiency of the filter with LCM and the one with SSL in the next section. This optimization has been proven in Coq and applied in our tool.

[6]It is the length of the longest busy window.

Algorithm 3 Dominated by Others

```
1: f denotes a workload function
2: l_f denotes a list of workload functions f_0, f_1, .., f_{n-1}
3: L denotes the sufficient length for determining the weak domination relations between workload functions from the list l_f
4: procedure DominatedByOthers(f, l_f, L)
5:     match l_f with
6:     | nil ⇒ false
7:     |f' :: l'_f ⇒
8:     % if f' is f itself (note that each function has an identifier)
9:     if f' is f then
10:        % then examine other functions l'_f
        return DominatedByOthers(f, l'_f, L)
11:        % else if f is dominated by f' return true
12:    else if Compare(f, f', L) then
        return true
13:    else
14:        % else examine other functions l'_f
        return DominatedByOthers(f, l'_f, L)
15:    end if
16: end procedure
```

Algorithm 4 Dominant Function Filter

```
1: l_f denotes a list of workload functions f_0, f_1, .., f_{n-1} to filter
2: L denotes the sufficient length for determining the weak domination relations between workload functions from the list l_f
3: procedure DominantFunction(l_1, l_2, L)
4:     match l_2 with
5:     | nil ⇒ l_1
6:     |wl :: l'_2 ⇒
7:     % if wl is dominated by any other function from l_1
8:     if DominatedByOthers(wl, l_1, L) then
9:         % then wl is removed from l_1 then the functions l'_2 are examined
        return DominantFunction(Filter(l_1, wl), l'_2, L)
10:    else
11:        % else examine the functions l'_2
        return DominantFunction(l_1, l'_2, L)
12:    end if
13: end procedure
14: % l_f^dom contains all dominant functions for the list l_f
15: l_f^dom := DominantFunction(l_f, l_f, L)
```

5.2 Avoiding recomputations

To analyze the response time of one task, we need to examine many scenarios, which are actually combinations of workload functions. And to certify a system, we need to analyze all tasks. Workload functions will be evaluated numerous times with the same arguments involving many recomputations. We improve this by applying techniques like memoization. For the sake of simplicity in proofs, we used lookup tables to store results of workload functions.

For each transaction, we can pre-calculate each workload function up to a large enough value[7], *e.g.,* its hyper-period HP and store all values in a table. When a specific value is needed, we search from this table, instead of recalculating. In addition, it is not necessary to compute all workload values for the domain $[0, HP[$. We only need to compute workloads for some specific durations *i.e.,* discontinued instants in Figure 5b when their workload values change. For instance, in Figure 5b, the instants when wl_1 must be computed are just 1, 2, 7, 11, 13, and similarly 1, 5, 7, 13, 14 for wl_2.

This optimization has been implemented and its correctness has been proven in Coq.

5.3 Heuristic algorithms

The analysis starts with approximate results, then it refines them by examining alignments transaction by transaction. But which transaction should be selected first to analyze? More generally, which order of transactions should be chosen by the refinement?

We investigated whether the order in which transactions are considered (*i.e.,* moved to SET_p) affects significantly the efficiency of the combined analysis and CertiCAN. For this, we implemented two different heuristics:

- Static order. We sort the transactions by putting the transactions with the highest utilization first. The intuition is that when computing an approximate workload, the higher the utilization of the transaction, the more pessimistic results probably are. In other words, if the transaction with the highest utilization is considered first for each refinement, the result is probably refined the most. Therefore, there will be more chances to speed up the analysis by using the utilization-sorted transaction list to refine. This configuration costs very little and works for most systems. This strategy is used by our tool. However, the gain of this optimization depends on the utilization distribution over transactions.
- Dynamic order. In the case where the utilization of the different transactions is similar, the static order does not provide significant benefit. In this case, we tried a dynamic order. For each refinement, we examine each transaction to refine (*i.e.,* compute the corresponding dominated scenarios as well as their WCRT) then choose the transaction with the largest WCRT to analyze. Thus, finding the transaction to pick at each step requires quite a lot more computations. In some cases, it speeds up the analysis, but slows it down for other cases. This optimization is not currently used in our tool.

6 Experimental Evaluation

Having completed the Coq formalization and correctness proofs of our analyses and their optimizations, we used the Coq extraction feature to obtain four certified tools: a precise analyzer[8],

[7] It should be greater than the length of any busy window computed in analyses. In our implementation, it is the length of the largest busy window that is computed when analyzing the lowest task. We have proven this is sufficient.

[8] Given that the precise analyzer has a very high time complexity and can only deal with small systems, it is not considered in the following experimentations which focus on large systems.

an approximate analyzer, a combined analyzer, and CertiCAN, the result certifier based on the combined analysis. Note that all these extracted analyzers integrate the optimizations presented in Section 5.

In this section, we evaluate these certified tools in terms of performance and scalability.

Table 1: Configuration parameters for NETCARBENCH generator.

ECUs	7 - 15
Utilization	40% - 60%
Period	$\{5, 10, 20, 50, 100, 200, 500, 1000\}$
Offset	random with granularity = 5
Priority	unique, arbitrary distribution
Transmission speed	500 kbits/s

The evaluated task sets are generated by NETCARBENCH[9], a benchmark generator for automotive message sets. This generator is used in the design and configuration of CAN and FlexRay communication systems. The following experimentations have been performed on 3000 systems that were generated by NETCARBENCH using a set of parameters presented in Table 1. More detail about configuration parameters[10] can be found in our NETCARBENCH configuration file in Appendix A. In all figures, all results are obtained from an Intel Core i7@2.6GHz, 16Gb, 64bits laptop.

Evaluation of analyzers

First, we compare the three analyzers: the approximate analyzer, the combined analyzer, and RTaW-Pegase.

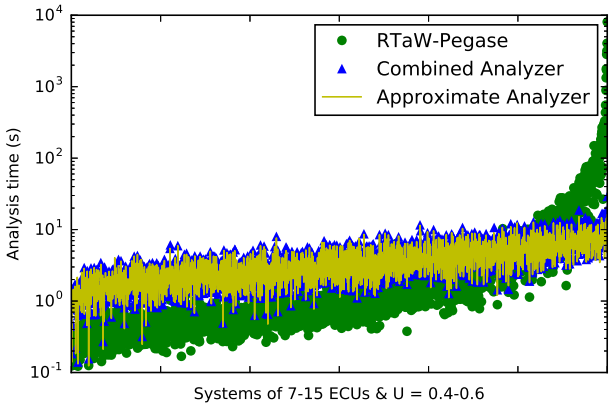


Figure 6: Comparison between certified analyzers and RTaW-Pegase

[9] http://www.netcarbench.org/

[10] These parameters have been provided by an expert from the automotive domain. Note that the utilization of the first ECU is allocated 30% of the whole utilization, *e.g.,* it is 18% if the system utilization is 60%.

Figure 6 shows that the combined analyzer has a remarkable performance. It returns precise results and its time efficiency is close to that of the approximate analyzer. Compared to Pegase, the combined analyzer has a better scalability. For the most complex systems, Pegase uses approximately two hours to compute a result whereas the combined analyzer needs less than 30 seconds to provide the same result. The main reason is that the combined analyzer combines two analyses (a precise and an approximate) in an optimized way that analyzes scenarios on demand. Other reasons may be that our optimizations avoid re-computations as much as possible, *e.g.,* by calculating the discontinued points and removing dominated scenarios. On the other hand, Pegase is more efficient than the combined analyzer on simple systems. One reason is that Pegase is written in C whereas the analyzer is extracted from Coq proofs in OCaml.

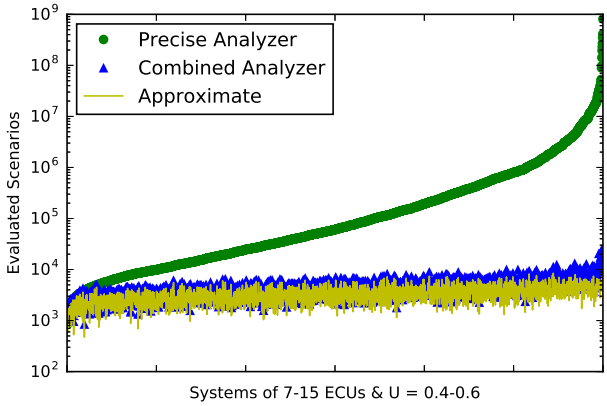


Figure 7: Evaluated scenarios by the certified analyzers

During our experimentations, we recorded the number of scenarios evaluated by the certified tools as showed in Figure 7. The number of scenarios for the precise analyzer is theoretically computed to be compared with the two other analyzers. This figure shows again that the combined analyzer has a good scalability. It is comparable to the approximate analyzer. Note that the trend of the combined and approximate analyzers' curves is similar to the one in Figure 6 because the runtime of the analyses depends directly on the number of scenarios that are evaluated.

CertiCAN *vs* Combined analyzer

We evaluate CertiCAN by verifying the results produced by the industrial tool RTaW-Pegase. We compared the performance of CertiCAN and the combined analyzer in Table 2.

The results in Table 2 show that CertiCAN is as good as the combined analyzer which is not surprising since they both share the same techniques and optimizations. Knowing the result to check, CertiCAN is a bit more efficient than the combined analyzer (by 17%). Both tools return a result in less than four seconds for most systems. For the most complex systems, they both only take less than half a minute. Note that the number of evaluated scenarios is not exactly proportional with the runtime because different scenarios may have a different time complexity, *e.g.,* an approximate scenario requires more computations to find the maximum workload among all its workload functions.

Table 2: Performance comparison between CertiCAN and the combined analyzer.

	Tool	**min**	**mean**	**median**	**max**
# Evaluated scenarios	CertiCAN	496	3573	3330	17136
	Combined Analyzer	855	4392	4082	25126
Runtime (s)	CertiCAN	0.10	**3.67**	3.09	24.29
	Combined Analyzer	0.14	**3.85**	3.23	29.15

Of course, CertiCAN is way more efficient than the combined analyzer for checking the schedulability of systems. We evaluated 100 more complex systems with the configuration (utilization = 60 - 80 %, 15 - 20 ECUs) presented in Appendix B. In this experiment, we verify the system schedulability using task deadlines as CertiCAN inputs. With this setting, CertiCAN is 45 times more efficient than the combined analyzer.

Impact of optimizations

We have evaluated 100 systems using the same configuration presented in Table 1 to understand the impact of applied optimizations using the same configuration presented in Table 2. The result is presented in Table 3. CertiCAN with 2 levels cannot deal with large systems, using many levels of refinement and a filter for filtering out dominated functions makes it possible. The filter with SSL is 5 - 8 times more efficient than the one with LCM. Avoiding recomputations[11] provides an improvement by a factor of 15 - 25.

Table 3: Impact of optimizations.

	Statistic	**2 levels**	Many Levels Filter-LCM Without AVO	Many Levels Filter-SSL Without AVO	Many Levels Filter-LCM With AVO	Many Levels Filter-SSL With AVO
CertiCAN runtime (s)	Mean	-	492	61	20	**4**
	Max	-	6435	329	831	**15**

According to all our experiments, we found that both the combined analyzer and CertiCAN have a high scalability. As far as we know, in modern cars, no more than 15-20 ECUs are connected to a single CAN bus [20, 21]. This indicates that CertiCAN can provide formal guarantees for industrial CAN bus analyzers.

It came as a surprise to find out that our analyzer was more efficient than Pegase for large systems. This is due to several sophisticated optimizations that we considered to improve the poor performance of the first versions of our tools. We now realize that all of them were not strictly necessary, at least to certify the results of Pegase. Integrating these optimizations into Pegase would make it more efficient than our certified tools, thus allowing it to analyze even larger systems.

7 Discussion

We have formally proven in Coq the correctness of the precise, approximate and generic RTAs presented in Section 3, the combined RTA and certifier of Section 4 and their optimizations of Section 7. Our proofs build upon the Prosa library [1] and use the basic definitions that

[11] In Table 3, AVO stands for avoiding recomputations, *i.e.,* lookup table, discontinued points.

it provides (task, job, arrival sequence, schedule, busy window, etc.). Note that if proofs are machine-checked, this cannot be the case for specifications and theorem declarations. Besides using the basic definitions from Prosa, we also had to define the FPNP scheduling policy and the task model. Compared to the proofs, those specifications are small and simple. They can be scrutinized and checked by the interested reader [2].

Table 4 illustrates the complexity of the proof effort (note that it excludes proofs from Prosa). Of course, formalizing these developments in Coq requires much more time and effort than on

Table 4: Proof effort for certifying CAN analyses.

Feature	**LOC**
System model (with proof)	1000
Workload property & removing re-computation	3142
Fixed point property	700
Busy window analysis	3037
Generic analysis	294
Combined analysis	1680
Combination property	545
Approximate and precise analyses	432
Candidate property	1562
Removing dominated candidates	1060
Analyzers & CertiCAN (with proof)	4000
Arithmetic proofs	1400
Total	18852

paper, but it also brings important benefits:

- It gives formal guarantees about the soundness of the specification and the absence of flaws in the proofs;
- It provides a better understanding of the role of each assumption, which helps to generalize proofs;
- The Coq extraction technique permits to produce formally verified tools (such as analyzers and certifiers) in the form of OCaml programs.

One of the most interesting by-products is that formalization often leads to more general and reusable proofs. For instance, our proof of busy window analysis does not rely on a specific task model but on abstract functions. It can be reused for other task models as long as we have the corresponding abstract functions.

Also, we defined two RTAs by instantiating the abstract functions with two different workload functions. Actually, the approach could be applied to get other RTAs with different levels of approximation. The combined analysis and certifier depend of generic properties (domination relations) that do not rely on the specific real-time model under study. The correctness proofs for the combined algorithm could apply to other kinds of analyses possibly disconnected from real-time theories.

8 Related work

To the best of our knowledge, CertiCAN is the first tool for certifying real-time systems analysis results. It is, however, built on top of existing results of real-time analyses, in particular formal proofs and abstraction refinement.

CertiCAN relies on formal definitions and lemmas from the Prosa library [1]. Prosa is the largest effort to date regarding the certification of real-time systems analyses. It is however not the first one. Previous publications in the area include [12], [13] and [17], based on the PVS proof assistant, which use state machines as the underlying formalism. The first two papers focus on the priority ceiling protocol and the latter on the scheduler of the Deos real-time operation system. While related to our work in a broader sense, these contributions do not tackle the problem of certifying RTA results.

Closer to us, a recent attempt [19] aims at certifying the results of Network Calculus computations using the interactive proof assistant Isabelle/HOL. In particular, [19] makes a case for result certification. The presented results are however preliminary and appear to have been discontinued. Our work can thus be seen as the concrete realization, with a different proof assistant and another underlying analysis technique, of the idea proposed in [19].

Our combined RTA follows a principle that is similar to the abstraction refinement method used in [22] and [23]. In particular, these two papers already use two different abstraction levels to compute precise bounds with increased efficiency. The main difference is that [22] deals with the analysis of digraph tasks with constrained deadlines, which does not fit the CAN context. We found this approach to be particularly well suited for result certification: having the response time bound to certify given as input makes the combined analysis all the more efficient.

9 Conclusion

In this paper, we have presented several RTA techniques and optimizations culminating in CertiCAN, a formally proven tool for the certification of CAN analysis results.

The analysis underlying CertiCAN is based on a combined use of two well-known CAN analysis techniques, one precise and the other approximate. The resulting analysis is as tight as the precise analysis but much faster. All analyses and certifier have been proven correct in Coq on top of the Prosa library. Thanks to the Coq extraction mechanism, we were able to produce the corresponding certified tools.

We have shown that the CertiCAN approach, which provides result certification rather than tool certification, is a realistic solution for industrial practice. The reason for this is twofold. First, it is flexible and light-weight in the sense that it does not depend on the internal structure or updates of the analysis tool that it complements. Second, it is efficient enough in terms of computation time. In particular, it is able to certify results computed by RTaW-Pegase, an industrial CAN analysis tool, even for large systems.

We believe that this work represents a significant step toward a formal certification of real-time systems analysis results in general. In particular, the underlying technique can be reused for any other system model for which there exist RTAs with different levels of precision. Future work includes the extension of the approach to networks of CAN buses and to other communication protocols.

Acknowledgment

This work has been partially supported by the LabEx PERSYVAL-Lab (grant ANR-11-LABX-0025-01) through the CASERM project and the French national research organization ANR (grants ANR-15-CE25-0008 and ANR-17-CE25-0016) through the VOCAL and RT-PROOFS projects. We would like to thank RealTime-at-Work for granting us an academic license and Jiajie Wang for providing an XML parser.

References

[1] A Library for formally proven schedulability analysis. http://prosa.mpi-sws.org/.

[2] A Tool for the Coq Certification of CAN Analysis Results. https://team.inria.fr/spades/certican-plus/.

[3] The CompCert C compiler. https://www.absint.com/compcert/.

[4] The Coq proof assistant. http://coq.inria.fr.

[5] The Isabelle proof assistant. https://isabelle.in.tum.de/.

[6] RTaW-Pegase: a Tool for Modeling, Simulation and automated Configuration of communication networks. http://www.realtimeatwork.com/software/rtaw-pegase/.

[7] The seL4 microkernel. https://sel4.systems/.

[8] K. Bletsas, N. C. Audsley, W. Huang, J. Chen, and G. Nelissen. Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. *LITES*, 5(1):02:1–02:20, 2018.

[9] Bosch. CAN specification version 2.0, 1991.

[10] F. Cerqueira, F. Stutz, and B. B. Brandenburg. Prosa: A case for readable mechanized schedulability analysis. In *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, pages 273–284. IEEE, 2016.

[11] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35(3):239–272, 2007.

[12] B. Dutertre. Formal analysis of the priority ceiling protocol. In *21st IEEE Real-Time Systems Symposium (RTSS)*, pages 151–160, 2000.

[13] B. Dutertre and V. Stavridou. Formal analysis for real-time scheduling. In *19th Digital Avionics Systems Conference (DASC)*, 2000.

[14] P. Fradet, X. Guo, J. Monin, and S. Quinton. Certican: A tool for the coq certification of CAN analysis results. In *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 182–191, 2019.

[15] P. Fradet, X. Guo, J.-F. Monin, and S. Quinton. A generalized digraph model for expressing dependencies. In *RTNS'18-26th International Conference on Real-Time Networks and Systems*, pages 1–11, 2018.

[16] X. Guo, S. Quinton, P. Fradet, and J.-F. Monin. Work-in-progress: Toward a coq-certified tool for the schedulability analysis of tasks with offsets. In *Real-Time Systems Symposium (RTSS), 2017 IEEE*, pages 387–389. IEEE, 2017.

[17] V. Ha, M. Rangarajan, D. Cofer, H. Rues, and B. Dutertre. Feature-based decomposition of inductive proofs applied to real-time avionics software: An experience report. In *26th International Conference on Software Engineering (ICSE)*, pages 304–313, 2004.

[18] X. Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[19] E. Mabille, M. Boyer, L. Fejoz, and S. Merz. Towards certifying network calculus. In *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, pages 484–489, 2013.

[20] A. Monot, N. Navet, B. Bavoux, and C. Maxim. Fine-grained simulation in the design of automotive communication systems. In *ERTSS-Embedded Real Time Software and Systems-2012*, 2012.

[21] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst. Typical worst case response-time analysis and its use in automotive network design. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*, pages 44:1–44:6, 2014.

[22] M. Stigge, N. Guan, and W. Yi. Refinement-based exact response-time analysis. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 143–152, 2014.

[23] M. Stigge and W. Yi. Combinatorial abstraction refinement for feasibility analysis of static priorities. *Real-Time Systems*, 51(6):639–674, 2015.

[24] K. Tindell. *Using offset information to analyse static priority pre-emptively scheduled task sets.* Technical report YCS 182. University of York, Department of Computer Science, 1992.

[25] K. Tindell. *Adding time-offsets to schedulability analysis.* University of York, Department of Computer Science, 1994.

[26] K. Tindell and A. Burns. Guaranteeing message latencies on controller area network (CAN). In *Proceedings of 1st international CAN conference*, pages 1–11, 1994.

[27] K. Tindell, A. Burns, and A. Wellings. Calculating controller area network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[28] K. Tindell, H. Hanssmon, and A. J. Wellings. Analysing real-time communications: Controller area network (CAN). In *Proceedings of the 15th IEEE Real-Time Systems Symposium (RTSS '94), San Juan, Puerto Rico, December 7-9, 1994*, pages 259–263, 1994.

[29] P. M. Yomsi, D. Bertrand, N. Navet, and R. I. Davis. Controller area network (can): Response time analysis with offsets. In *2012 9th IEEE International Workshop on Factory Communication Systems*, pages 43–52, May 2012.

Appendices

A Configuration file 1 for NETCARBENCH

```
<netcarbench-data version="3.2" >
<can-network name="EVALUATION" granularity="5" bandwidth="500" >
<network-load min="0.4" max="0.6" />
<nb-network-interfaces min="7" max="15" />
<fixed-station-loads>
<station id="1" value="0.30" />
</fixed-station-loads>
<frame-periods>
<period value="5" weight="2" margin="1" prio_low_range="1" prio_high_range="200"/>
<period value="10" weight="5" margin="2" prio_low_range="201" prio_high_range="400"/>
<period value="20" weight="5" margin="2" prio_low_range="401" prio_high_range="600"/>
<period value="50" weight="10" margin="4" prio_low_range="601" prio_high_range="800"/>
<period value="100" weight="10" margin="4" prio_low_range="801" prio_high_range="1000"/>
<period value="200" weight="5" margin="2" prio_low_range="1001" prio_high_range="1200"/>
<period value="500" weight="2" margin="1" prio_low_range="1201" prio_high_range="1400"/>
<period value="1000" weight="2" margin="1" prio_low_range="1401" prio_high_range="1600"/>
</frame-periods>
<frame-payloads>
<payload value="1" weight="1" margin="1"/>
<payload value="2" weight="1" margin="1"/>
<payload value="3" weight="1" margin="1"/>
<payload value="4" weight="2" margin="1"/>
<payload value="5" weight="3" margin="2"/>
<payload value="6" weight="4" margin="2"/>
<payload value="7" weight="5" margin="2"/>
<payload value="8" weight="6" margin="3"/>
</frame-payloads>
<offsets mode="RANDOM"/>
</can-network>
</netcarbench-data>
```

Listing 1: Configuration file 1 for NETCARBENCH

B Configuration file 2 for NETCARBENCH

```
<netcarbench-data version="3.2" >
<can-network name="EVALUATION" granularity="5" bandwidth="500" >
<network-load min="0.6" max="0.8" />
<nb-network-interfaces min="15" max="20" />
<fixed-station-loads>
<station id="1" value="0.30" />
</fixed-station-loads>
<frame-periods>
<period value="5" weight="2" margin="1" prio_low_range="1" prio_high_range="200"/>
<period value="10" weight="5" margin="2" prio_low_range="201" prio_high_range="400"/>
<period value="20" weight="5" margin="2" prio_low_range="401" prio_high_range="600"/>
<period value="50" weight="10" margin="4" prio_low_range="601" prio_high_range="800"/>
<period value="100" weight="10" margin="4" prio_low_range="801" prio_high_range="1000"/>
<period value="200" weight="5" margin="2" prio_low_range="1001" prio_high_range="1200"/>
<period value="500" weight="2" margin="1" prio_low_range="1201" prio_high_range="1400"/>
<period value="1000" weight="2" margin="1" prio_low_range="1401" prio_high_range="1600"/>
</frame-periods>
<frame-payloads>
<payload value="1" weight="1" margin="1"/>
<payload value="2" weight="1" margin="1"/>
<payload value="3" weight="1" margin="1"/>
<payload value="4" weight="2" margin="1"/>
<payload value="5" weight="3" margin="2"/>
<payload value="6" weight="4" margin="2"/>
<payload value="7" weight="5" margin="2"/>
<payload value="8" weight="6" margin="3"/>
</frame-payloads>
<offsets mode="RANDOM"/>
</can-network>
</netcarbench-data>
```

Listing 2: Configuration file 2 for NETCARBENCH

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399