CertiCAN
Certifying CAN Analyses and Their Results

Pascal Fradet[1*], Xiaojie Guo[1,2*] and Sophie Quinton[1*]

[1]Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LIG.
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG
RealTime-at-Work, France.
[*]Inria Rhône-Alpes, 655 av. de l'Europe, 38330 Montbonnot, France.

*Corresponding author(s). E-mail(s): pascal.fradet@inria.fr;
xiaojie.guo@realtimeatwork.com; sophie.quinton@inria.fr;

Abstract

We present CertiCAN, a tool produced using the Coq proof assistant and based on the Prosa library for the formal verification of CAN analysis results. Result verification is a process that is lightweight and flexible compared to tool verification. Indeed, the formal verification 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 faster to check a result than to produce it. All these reasons make CertiCAN a practical choice for industrial purposes. CertiCAN is based on the formal verification 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.

1 Introduction

There is a general trend toward formally verified[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 (Bletsas et al, 2018) 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 and Burns (1994) and Tindell et al (1994, 1995), which was found and fixed only many years later by Davis et al (2007). This motivated the development of Prosa (Cerqueira et al, 2016), an open-source library of definitions and proofs for real-time systems analysis based on the Coq proof assistant, which has become a reference and that we use in our work. 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 prove correct complex applications – see, for example, the use of the CompCert C compiler (Leroy, 2009) based on Coq or the Sel4 microkernel based on Isabelle (Nipkow et al, 2002). 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 formal 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 protocol (Bosch, 1991) is widely used in automotive applications and there exist several commercial tools performing CAN analysis. Among these, we focus on RTaW-Pegase, for which we obtained an academic license.

Rather than verifying 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 and the Prosa library that can formally verify 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 formal result verification is a process that is lightweight and flexible compared to formal tool verification, which makes it a practical choice for industrial purposes. Indeed, RTaW-Pegase is a complex tool for which we do not have the

[1] Within the proof assistants community, the term *certified* is commonly used to refer to formally verified programs and software. Throughout this paper, we mostly use *formally verified* instead.

[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.

source code. It is likely to integrate complex optimizations and to be subject to regular updates. All this would make it difficult to formally verify RTaW-Pegase itself and would require to update the correctness proof regularly.

Our problem is then: Can we verify 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 verify?

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 (Tindell, 1992, 1994). The precise analysis was adapted to CAN by Yomsi et al (2012).

CertiCAN combines in an optimized way the two analyses. It first tries to verify the result provided as input using the approximate analysis only, then resorts to a more precise analysis for cases which cannot be verified 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 precisions, *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-verified tools extracted in OCaml from the proofs, one for each analysis;
5. based on the same principle as the new RTA, a method and its corresponding verified tool (called CertiCAN) to certify the results of non-verified 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 formal result verification 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 (CertiCAN).

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.

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 verifying 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 Guo et al (2017) and Fradet et al (2019). Since then, CertiCAN has been refined and greatly improved using novel and proved correct 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. The Coq sources and proof scripts are available online CertiCAN (2021). The interested reader can examine, check and run them to reproduce our experiments.

2 Context and System Model

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, using a notion of transaction to account for the fact that only messages sent from the same Electronic Control Unit (ECU) are synchronized (Yomsi et al, 2012). In addition to the precise analysis, RTaW-Pegase proposes an 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 (Bosch, 1991) 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 The 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);

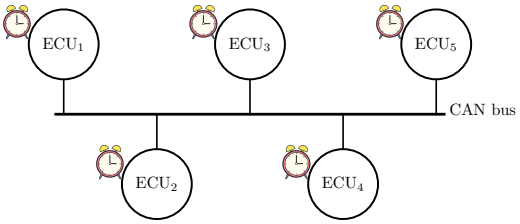

Figure 1: An example of CAN bus

- 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 (Davis et al, 2007).

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 the message missing its deadline. To address this issue, message activations are separated by introducing an *offset* for each message. An offset is a fixed time duration from local time 0 to the first activation time of a message. Adding offsets allows shifting periods to increase bus utilization while maintaining system schedulability.

2.2 System model

The system model 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):

$$T_i := \{\tau_{i,k_1}, \tau_{i,k_2}, \ldots, \tau_{i,k_M}\}$$

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.,* $o_{i,k} < p_{i,k}$.

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. The model however 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. Prosa, and therefore our model, is based on discrete time and does not easily support the representation of clock drifts. The timing uncertainty due to different clocks is expressed by *transaction offsets*: each transaction T_i is activated periodically after an unknown duration o_i called transaction offset. This representation is sufficient if the clock drifts are negligible for short time intervals (*i.e.,* within a single busy window). In the rest of the paper, we use the term "offset" to refer to a "task offset". Whenever we mean "transition offset" we explicitly mention it this way. Note that task offsets are part of the system model while transaction offsets may vary from one execution to the next.

Each transaction T_i is then activated again with a period (called the hyperperiod) equal to the least common multiple of the periods of all its tasks. Formally,

$$HP_i = lcm\{p_{i,k_1}, p_{i,k_2}, \ldots, p_{i,k_M}\}$$

Task $\tau_{i,k}$ activates its jobs periodically at $o_i + o_{i,k} + m \times p_{i,k}$ 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 $c(\jmath) \leq c^+_{i,k}$

Jobs are scheduled over the CAN bus in a fixed-priority nonpreemptive (FPNP) manner: job execution cannot be interrupted once started, and the scheduler always chooses the job that has the highest priority amongst the jobs that are ready for execution. The response time $r(\jmath)$ of job $\jmath$ 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 whose priorities are higher than or equal to, higher than or lower than k, respectively.

3 Certified Response Time Analyses for CAN

In this section, we describe two standard RTAs for CAN, a precise analysis and an approximate one, that will form the basis of our own combined analysis and result certifier. The correctness of these RTAs has been proven using the Coq proof assistant on top of the Prosa library. The presentation mostly follows the Coq specification, with a few reformulations for readability. We omit proofs of lemmas and theorems, focusing on providing some intuition, and refer the interested reader to the Coq source (CertiCAN).

The precise analysis proceeds by considering all possible alignments between transactions in its response time bound computation. The approximate analysis examines a much smaller set of approximate scenarios where all transactions, except the one of the task under study, are abstracted by an approximate workload function. 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. Both analyses rely heavily on the well-known concept of busy window, that we explain now.

3.1 Busy window analysis

Let $\tau_{i,k}$ be a task for which we want to bound the worst-case response time. Throughout this section, i and k explicitly refer to this task.

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 bw^k)**.** *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] *$[t_1, t_2[$.*

[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.

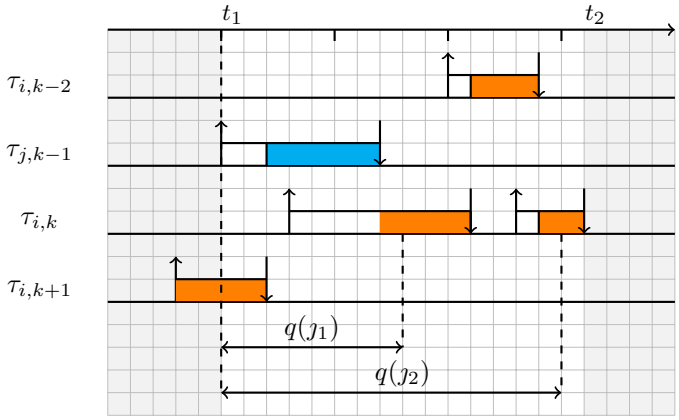

Figure 2: An illustrative level-k busy window.

Figure 2 shows an example of such a level-k busy window $[t_1, t_2[$. Note that, due to the non-preemptive nature of scheduling, a job with a 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 $b_k(t_1)$. In our example, $b_k(t_1)$ is equal to 2.

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] From now on, in this section, t_1 and t_2 explicitly refer to this specific busy window.

Clearly, a job with a priority higher than or equal to k completes in the level-k busy window in which it is activated. Therefore, $\jmath$'s 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 Fradet et al (2018).

Definition 3 (Queueing delay)**.** *The* queueing delay *of $\jmath$, 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 $\tau_{i,k}$ in the represented level-k busy window, where $\jmath_n$ denotes the n-th job of task $\tau_{i,k}$ activated in that busy window.

[4] It can be shown that if the utilization is below 100%, it is always possible to compute such a level-k busy window. We do not detail the proof here but the interested reader can find the Coq proof in the online artifact.

Definition 4 (Phase)**.** *The* phase *of $\jmath$, denoted $\varphi(\jmath)$, is the duration $act(\jmath) - t_1$,* i.e., *the duration between t_1 and the activation time of that job.*

For instance, in Figure 2, the phase of $\jmath_1$ is 3, and that of $\jmath_2$ is $3+1\times10 = 13$.

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 an upper bound on $r(\jmath)$. In fact, we are going to simultaneously compute response time bounds of all jobs of $\tau_{i,k}$ in the same busy window as $\jmath$. We will thus use the following notations:

- $n(\jmath)$ denotes the number of jobs of task $\tau_{i,k}$ in busy window $[t_1, t_2[$;
- $\jmath_n$ denotes the n-th job of $\tau_{i,k}$ in busy window $[t_1, t_2[$ for $n \leq n(\jmath)$.

Lemma 1.

$$r(\jmath) \leq \max_{n \leq n(\jmath)} \{q(\jmath_n) - 1 + c^+_{i,k} - \varphi(\jmath_n)\} \tag{2}$$

Note that $c(\jmath)$ is trivially bounded by $c^+_{i,k}$.

Our goal is now to provide an upper bound on $n(\jmath)$, and then an upper bound on $q(\jmath_n)$ and a lower bound on $\varphi(\jmath_n)$ for any given $n \leq n(\jmath)$. For that, we will use the notion of workload.

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

Notation 1. *Let $next_{i,k}(t_1)$ denote the time of the first activation of task $\tau_{i,k}$ after t_1, which could correspond to $\jmath$ or another job in the same busy window.*

Lemma 2.

$$n(\jmath) = \left\lceil \frac{t_2 - 1 - next_{i,k}(t_1)}{p_{i,k}} \right\rceil \tag{3}$$

In addition, one only needs to know t_1, and not t_2, to compute $n(\jmath)$. Indeed, t_2 can be deduced from the duration of the busy window during which $\jmath$ executes,

which can be found by computing the least fixed point of the following equation:

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

with

$$f_{bw}^{\jmath}(\Delta) := b_k(t_1) + \sum_{\tau_{j,l} \in hep(k)} wl_{j,l}(t_1, \Delta) \tag{4}$$

This result is a direct consequence of Definitions 1 and 2: a busy window closes at the earliest time $t_2 := t_1 + \Delta$ such that all jobs of priority higher than or equal to k activated strictly before t_2 have completed at t_2. The queueing delay of $\jmath_n$ for any $n \leq n(\jmath)$ can be found similarly.

Lemma 3. *For any $n \leq n(\jmath)$, the queueing delay of $\jmath_n$ is the least fixed point of the following equation:*

$$\Delta = f_q^{\jmath}(n, \Delta)$$

where

$$f_q^{\jmath}(n, \Delta) := b_k(t_1) + \sum_{\tau_{j,l} \in hp(k)} wl_{j,l}(t_1, \Delta) + wl_{i,k}\left(t_1, \varphi(\jmath_n)\right) + 1 \tag{5}$$

with

$$\varphi(\jmath_n) := next_{i,k}(t_1) - t_1 + (n-1) \times p_{i,k} \tag{6}$$

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

The key to upper bounding $n(\jmath)$ is to upper bound $f_{bw}^{\jmath}(\Delta)$ by a function $f_{bw}^{+}(\Delta)$ that does not depend on $\jmath$, and then use the following lemma to conclude that the least fixed point of $f_{bw}^{+}(\Delta)$ is an upper bound on the least fixed point of $f_{bw}^{\jmath}(\Delta)$.

Lemma 4. *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. If (1) for all $x : \mathbb{N}$, $f(x) \leq g(x)$ and (2) for all $x : \mathbb{N}^+$ such that $x < \Delta_1$, we have $x < f(x)$, then $\Delta_1 \leq \Delta_2$.*

The same process applies to $q(\jmath_n)$: if we upper bound $f_q^{\jmath}(n, \Delta)$ by a function $f_q^{+}(n, \Delta)$, then we can use Lemma 4 to conclude that the least fixed point of $f_q^{+}(n, \Delta)$ is an upper bound on $q(\jmath_n)$. Based on that, we can use $f_{bw}^{+}(\Delta)$ and $f_q^{+}(n, \Delta)$ to come up with a bound r^+ on $r(\jmath)$.

The principle of the response time analyses that will be presented in the next sections is slightly more elaborate: instead of using one function $f_{bw}^{+}(\Delta)$

and one $f_q^+(n, \Delta)$ to upper bound $f_{bw}^{\jmath}(\Delta)$ and $f_q^{\jmath}(n, \Delta)$ for all jobs, the precise, approximate and generic RTAs use a finite set of *scenarios* and corresponding functions such that there is always one scenario s for which $r(\jmath) \leq r^+(s)$. The WCRT of task $\tau_{i,k}$ is thus found by taking the maximum WCRT found for all these scenarios. As the next two sections show, the workload bounds are thus computed differently for the precise analysis and the approximate analysis.

3.2 Precise RTA

Using the same notations as in the previous section, let $\jmath$ be an arbitrary job of task $\tau_{i,k}$, executing within a level-k busy window denoted $[t_1, t_2[$. We will show in this section that there is a scenario such that

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

from which we can obtain an upper bound on $n(\jmath)$ and then, for any $n \leq n(\jmath)$, an upper bound on $q(\jmath_n)$ and a lower bound on $\varphi(\jmath_n)$. In practice, the major part of this section is dedicated to providing, for any transaction T_j, an upper bound for $wl_{j,l}(t_1, \Delta)$ that only depends on scenarios as described above. From that, we will then derive upper bounds on the other terms, all the way to the response time of $\jmath$.

Let us start by upper bounding the blocking delay.

Lemma 5 (Blocking delay bound)**.** $b_k(t_1) \leq b_k^+$ *where*

$$b_k^+ := \max_{\tau_{j,l} \in lp(k)} (c_{j,l}^+ - 1) \tag{7}$$

Indeed, blocking is maximized if 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. Note that this upper bound may introduce pessimism in the response time bound computed by the precise analysis. 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.

Lemma 6 (Workload bound)**.** *For any instant t and duration Δ, $wl_{j,l}(t, \Delta)$ is bounded by*

$$wl_{j,l}^+(t, \Delta) := \left\lceil \frac{t + \Delta - next_{j,l}(t)}{p_{j,l}} \right\rceil \times c_{j,l}^+ \tag{8}$$

with $next_{j,l}(t)$ denoting the instant of the first activation of task $\tau_{j,l}$ after t.

We now provide an upper bound for $wl_{j,l}^+(t_1, \Delta)$.

Notation 2. *Let $next_j(t_1)$ denote the first activation after t_1 of any task in $T_j \cap hep(k)$.*

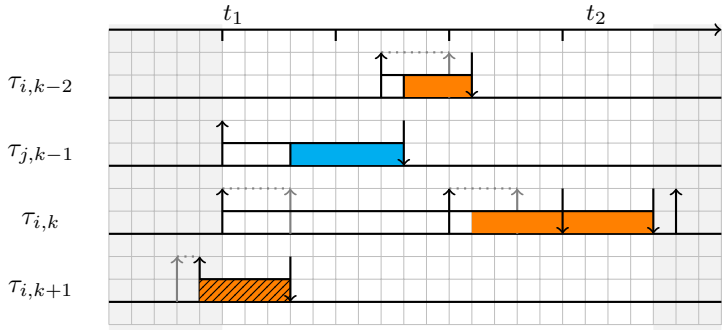

Figure 3: The scenario corresponding to the level-k busy window of Figure 2.

Lemma 7. $wl^+_{j,l}(t_1, \Delta) \leq wl^+_{j,l}(next_j(t_1), \Delta)$

The intuition behind this result is that the workload bound of any task in T_j can only increase if we shift all activations of T_j to align t_1 with $next_j(t_1)$. Figure 3 illustrates this: Transaction T_j is already aligned with t_1, that is, $next_j(t_1) = t_1$. For transaction T_i however, $next_i(t_1) = t_1 + 3$. Transaction T_i is thus shifted to the left to align with t_1 its first activation of a job in $hep(k)$ in the busy window. Note that the low priority task of T_i is in contrast shifted to the right to maximize blocking. This situation is not possible in practice since $\tau_{i,k-1}$ is synchronized with the other tasks in T_i, so the computed bound may be pessimistic.

The key element of the precise analysis is that, no matter what the value of t_1, $wl^+_{j,l}(next_j(t_1), \Delta)$ can only take a finite number of values for a given Δ: at the end of the hyperperiod of T_j, the activation pattern, and therefore the workload, repeats.

Definition 6 (Alignment)**.** *The* alignment *of transaction* T_j w.r.t. t_1*, denoted* $al_j(t_1)$*, is the duration between the latest activation of transaction* T_j *before* t_1*, and* $next_j(t_1)$*.*

We call $al_j(t_1)$ an alignment because it identifies which activation[5] of T_j in the hyperperiod HP_j (say, the x-th) is aligned with t_1 if we shift all activations of T_j. Note that $al_j(t_1)$ is an element of Al_j defined as follows.

$$Al_j := \bigcup_{\tau_{j,l} \in T_j \cap hep(k)} \{\, al_j = o_{j,l} + m \times p_{j,l} \mid m \in \mathbb{N} \wedge al_j < HP_j \,\} \tag{9}$$

[5]Note that there may be several activations at the same instant, which does not alter the reasoning.

Each element in Al_j corresponds to the activation of a job with a priority higher than or equal to k in transaction T_j within its hyper-period HP_j. This is illustrated (on a different example) in Figure 4.

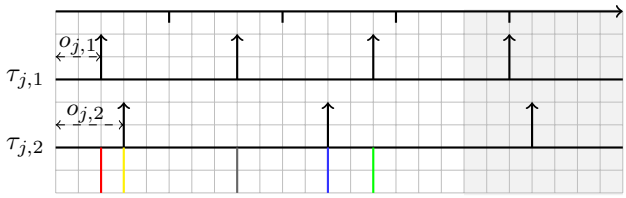

Figure 4: A transaction with two tasks. Their offsets are 2 and 3, their periods are 6 and 9, respectively. The LCM of task periods is 18, during which there will be 5 activations. This transaction, therefore, represents 5 alignments. Each vertical colored line indicates an alignment.

Interestingly, the number of activations of $\tau_{j,l}$ between $next_j(t_1)$ and $next_j(t_1) + \Delta$ is the same as the number of activations of $\tau_{j,l}$ between $o_j + al_j(t_1)$, *i.e.,* the instant of the x-th activation of T_j in its first hyperperiod, and $o_j + al_j(t_1) + \Delta$. Hence the following lemma.

Lemma 8. $wl^+_{j,l}(next_j(t_1), \Delta) = wl^+_{j,l}(o_j + al_j(t_1), \Delta)$

Note that in fact, $wl^+_{j,l}(o_j + al_j(t_1), \Delta)$ does not depend on the size of o_j, as can be easily derived from the definition of $wl^+_{j,l}$ given by Equation 8. While we keep transaction offsets explicit in the paper for readability, we have in our Coq proof introduced an intermediate notation to get rid of o_j.

The consequence of all this 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.

The alignments $al_j(t_1)$ *w.r.t.* all transactions T_j form the precise scenario $s_\jmath$ which we will use to upper bound the response time of all jobs of $\tau_{i,k}$ in $[t_1, t_2[$.

$$s_\jmath := (al_1(t_1), \cdots, al_N(t_1)) \tag{10}$$

We have now everything we need to provide an upper bound on $n(\jmath)$.

Lemma 9. *For any time duration* Δ, $f^\jmath_{bw}(\Delta) \leq f^+_{bw}(s_\jmath, \Delta)$, *where*

$$f^+_{bw}(s_\jmath, \Delta) := b^+_k + \sum_{\tau_{j,l} \in hep(k)} wl^+_{j,l}(o_j + al_j(t_1), \Delta) \tag{11}$$

Lemma 10. *Let* $bw^+(s_\jmath)$ *denote the least fixed point of* $f^+_{bw}(s_\jmath, \Delta)$. *Then* $n(\jmath) \leq n^+(s_\jmath)$ *where*

$$n^+(s_\jmath) := \left\lceil \frac{bw^+(s_\jmath)}{p_{i,k}} \right\rceil \tag{12}$$

Let us now provide an upper bound on the response time of $\jmath_n$ for any $n \leq n(\jmath)$. From Lemma 1, we know that this can be achieved by upper bounding $q(\jmath_n)$ and lower bounding $\varphi(\jmath_n)$. We start with the latter, and show that one can lower bound $\varphi(\jmath_n)$ with a function of $al_i(t_1)$.

Lemma 11. $\varphi(\jmath_n) \geq \varphi_n(al_i(t_1))$ *where*

$$\varphi_n(al_i(t_1)) := o_{i,k} - al_i(t_1) + (n-1) \times p_{i,k} \tag{13}$$

This is a direct consequence of Equation 6 and of the fact that $o_{i,k} - al_i(t_1) = next_{i,k}(t_1) - next_i(t_1)$ and $next_i(t_1) \geq t_1$.

Lemma 12. *For any duration* Δ, $f^\jmath_q(n, \Delta) \leq f^+_q(n, s_\jmath, \Delta)$ *where*

$$f^+_q(n, s_\jmath, \Delta) := b^+_k + \sum_{\tau_{j,l} \in hp(k)} wl^+_{j,l}(o_j + al_j(t_1), \Delta) + (n-1) \times c^+_{i,k} + 1 \tag{14}$$

This is a direct consequence of Equation 5 and Lemmas 5 to 8. Note that $wl_{i,k}(t_1, \varphi(\jmath_n))$ is upper bounded by $(n-1) \times c^+_{i,k}$: There are $n-1$ jobs of $\tau_{i,k}$ to be executed before $\jmath_n$ in the busy window under study, and they will each take at most $c^+_{i,k}$ to complete their execution.

Lemma 13. $r(\jmath) \leq r^+(s_\jmath)$ *where*

$$r^+(s_\jmath) := \max_{n \leq n^+(s_\jmath)} \{q^+_n(s_\jmath) - 1 + c^+_{i,k} - \varphi_n(al_i)\} \tag{15}$$

with $q^+_n(s_\jmath)$ *denoting the least fixed point of* $f^+_q(n, s_\jmath, \Delta)$.

To upper bound the WCRT of $\tau_{i,k}$, we now need to analyze all possible alignments al_j for each transaction T_j, representing all possible alignments of the busy window with an activation.

Definition 7 (Precise scenario)**.** *A precise scenario* s *is a tuple* $(al_1, \cdots, al_N)$ *specifying one alignment for each transaction, where* $al_j \in Al_j$ *for each* $1 \leq j \leq N$. *We denote the set of all precise scenarios* $\mathcal{S} := Al_1 \times \cdots \times Al_N$.

The set of precise scenarios consists of all combinations of alignments over all transactions. The worst-case response time $wcrt_{i,k}$ of task $\tau_{i,k}$ is thus bounded by the maximum WCRT of all scenarios.

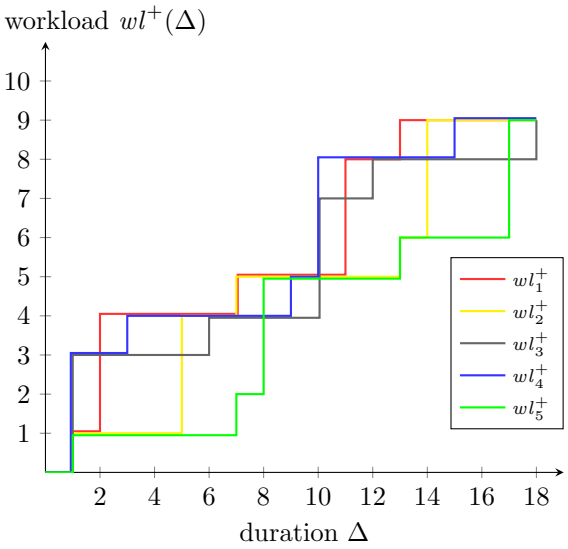

Figure 5: Workload functions corresponding to the five alignments of the transaction of Figure 4, under the assumption that the worst-case execution time of $\tau_{i,1}$ is 1 and that of $\tau_{i,2}$ is 3.

Theorem 1 (Precise analysis)**.** $wcrt_{i,k} \leq \max_{s \in \mathcal{S}}\{r^+(s)\}$

3.3 Approximate RTA

For large systems, the number of precise scenarios explodes and the precise analysis quickly becomes intractable. In this section, we present a more efficient but approximate analysis, following the approach presented in Tindell (1992). Its principle is to use approximate scenarios, which consist of 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 bound among all possible alignments represented by Al_j:*

$$wl_j^*(\Delta) := \max_{al_j \in Al_j} \left\{ \sum_{\tau_{j,l} \in\, T_j \cap hep(k)} wl_{j,l}^+(o_j + al_j, \Delta) \right\} \tag{16}$$

As a example, the approximate workload bound function of the transaction presented in Figure 4 would be obtained by taking the maximum over the five workload functions corresponding to the five possible alignments, as shown in Figure 5.

In the approximate analysis, we upper bound the workload of each transaction T_j using $wl_j^*(\Delta)$, except for T_i for which we use the precise workload bound. With this strategy, we compute new bounds for the busy window length and queueing delay, which only depend on the alignment of transaction T_i.

Lemma 14. *For any precise scenario* $s = (al_1, \cdots, al_N) \in \mathcal{S}$ *and duration* Δ, $f_{bw}^+(s, \Delta) \leq f_{bw}^*(al_i, \Delta)$ *where*

$$f_{bw}^*(al_i, \Delta) := b_k^+ + \sum_{\substack{T_j \in Sys \\ j \neq i}} wl_j^*(\Delta) + \sum_{\tau_{i,l} \in T_i \cap hep(k)} wl_{i,l}^+(o_i + al_i, \Delta) \tag{17}$$

Lemma 15. *For any precise scenario* $s = (al_1, \cdots, al_N) \in \mathcal{S}$, $n^+(s) \leq n^*(al_i)$ *where*

$$n^*(al_i) := \left\lceil \frac{bw^*(al_i)}{p_{i,k}} \right\rceil \tag{18}$$

with $bw^*(al_i)$ *denoting the least fixed point of* $f_{bw}^*(al_i, \Delta)$.

Lemma 16. *For any precise scenario* $s = (al_1, \cdots, al_N) \in \mathcal{S}$ *and duration* Δ, $f_q^+(n, s, \Delta) \leq f_q^*(n, al_i, \Delta)$ *where*

$$f_q^*(n, al_i, \Delta) := b_k^+ + \sum_{\substack{T_j \in Sys \\ j \neq i}} wl_j^*(\Delta) + \sum_{\tau_{i,l} \in T_i \cap hp(k)} wl_{i,l}^+(o_i + al_i, \Delta) + (n-1) \times c_{i,k}^+ + 1 \tag{19}$$

Lemma 17. *For any precise scenario* $s = (al_1, \cdots, al_N) \in \mathcal{S}$, $r^+(s) \leq r^*(al_i)$ *where*

$$r^*(al_i) := \max_{n \leq n^*(al_i)} \{ q_n^*(al_i) - 1 + c_{i,k}^+ - \varphi_n(al_i) \} \tag{20}$$

with $q_n^*(al_i)$ *denoting the least fixed point of* $f_q^*(n, al_i, \Delta)$.

Compared to the precise analysis, the approximate analysis does not consider all possible combinations (the cartesian product) of all alignments of all transactions, but only the possible alignments of transaction T_i. In other words, the set of approximate scenarios is equal to Al_i.

Theorem 2 (Approximate analysis)**.** $wcrt_{i,k} \leq \max_{al_i \in Al_i} \{r^*(al_i)\}$

3.4 Generic RTA

We have presented two response time 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.

The fundamental difference between the two analyses is the set of scenarios that are analyzed to upper bound the response time. We now present a generic analysis which allows intermediate levels of approximation in the definition of scenarios by explicitly specifying the transactions for which the approximate workload bound function will not be used. The generic analysis is thus parameterized by a set of transactions $\mathcal{T} \subseteq Sys$ such that $T_i \in \mathcal{T}$. If $\mathcal{T} = Sys$ then no workload bound will be approximated and the generic analysis is equivalent to the precise analysis. The other extreme is if $\mathcal{T} = \{T_i\}$, in which case the workload bound of all transactions except T_i will be approximated, which corresponds to the approximate analysis we presented in the previous section.

Definition 9 (Generic scenario)**.** *Given a set $\mathcal{T} \subseteq Sys$ such that $T_i \in \mathcal{T}$, a generic scenario* w.r.t. *$\mathcal{T}$ is a tuple specifying one alignment $al_j \in Al_j$ for each transaction T_j in $\mathcal{T}$. For any precise scenario $s = (al_1, \cdots, al_N) \in \mathcal{S}$, we denote $s|_{\mathcal{T}}$ the generic scenario* w.r.t. *$\mathcal{T}$ that is obtained by keeping only the alignments of s that correspond to transactions in $\mathcal{T}$. We denote $\mathcal{S}|_{\mathcal{T}}$ the set of all generic scenarios* w.r.t. *$\mathcal{T}$, formally: $\mathcal{S}|_{\mathcal{T}} := \Pi_{T_j \in \mathcal{T}} Al_j$*

Suppose given a set $\mathcal{T} \subseteq Sys$ such that $T_i \in \mathcal{T}$. To present the generic analysis, we proceed as for the approximate analysis, except we only use the approximate workload bound for transactions in $Sys \backslash \mathcal{T}$.

Lemma 18. *For any precise scenario $s = (al_1, \cdots, al_N) \in \mathcal{S}$ and duration Δ, $f^+_{bw}(s, \Delta) \leq f^\star_{bw}(s|_{\mathcal{T}}, \Delta)$ where*

$$f^\star_{bw}(s|_{\mathcal{T}}, \Delta) := b^+_k + \sum_{T_j \in Sys \backslash \mathcal{T}} wl^*_j(\Delta) + \sum_{\substack{\tau_{j,l} \in T_j \cap hep(k) \\ T_j \in \mathcal{T}}} wl^+_{j,l}(o_j + al_j, \Delta) \tag{21}$$

Lemma 19. *For any precise scenario $s = (al_1, \cdots, al_N) \in \mathcal{S}$, $n^+(s) \leq n^\star(s|_{\mathcal{T}})$ where*

$$n^\star(s|_{\mathcal{T}}) := \left\lceil \frac{bw^\star(s|_{\mathcal{T}})}{p_{i,k}} \right\rceil \tag{22}$$

with $bw^\star(s|_{\mathcal{T}})$ denoting the least fixed point of $f^\star_{bw}(s|_{\mathcal{T}}, \Delta)$.

Lemma 20. *For any precise scenario $s = (al_1, \cdots, al_N) \in \mathcal{S}$ and duration Δ, $f^+_q(n, s, \Delta) \leq f^\star_q(n, s|_{\mathcal{T}}, \Delta)$ where*

$$f^\star_q(n, s|_{\mathcal{T}}, \Delta) := b^+_k + \sum_{T_j \in Sys \backslash \mathcal{T}} wl^*_j(\Delta) + \sum_{\substack{\tau_{j,l} \in T_j \cap hp(k) \\ T_j \in \mathcal{T}}} wl^+_{j,l}(o_j + al_j, \Delta) + (n-1) \times c^+_{i,k} + 1 \tag{23}$$

Lemma 21. *For any precise scenario* $s = (al_1, \cdots, al_N) \in \mathcal{S}$, $r^+(s) \leq r^\star(s|_\mathcal{T})$ *where*

$$r^\star(s|_\mathcal{T}) := \max_{n \leq n^\star(s|_\mathcal{T})} \{ q_n^\star(s|_\mathcal{T}) - 1 + c_{i,k}^+ - \varphi_n(al_i) \} \tag{24}$$

with $q_n^\star(s|_\mathcal{T})$ *denoting the least fixed point of* $f_q^\star(n, s|_\mathcal{T}, \Delta)$.

Theorem 3. $wcrt_{i,k} \leq \max_{s \in \mathcal{S}|_\mathcal{T}} r^\star(s)$

Note that our theorem holds for any set $\mathcal{T} \subseteq Sys$ such that $T_i \in \mathcal{T}$. The time complexity of any instance of the generic analysis (determined by the choice of $\mathcal{T}$) relates to its number of scenarios (*i.e.,* the size of $\mathcal{S}|_\mathcal{T}$) and lies between the complexity of the precise analysis and that of the approximate one — which are both instances of the generic analysis. The same holds for the precision of the analysis. This makes it possible to choose different trade-offs between precision and time complexity.

4 Combined RTA and its 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 our 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. We then 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.

Let us start with a few remarks that relate the precise and the approximate analyses. The worst-case response time is upper bounded using the precise analysis by $\max_{s \in \mathcal{S}}\{r^+(s)\}$ while the upper bound computed by the approximate analysis is $\max_{al_i \in Al_i}\{r^*(al_i)\}$. Now remember Lemma 17: for any precise scenario $s = (al_1, \cdots, al_N) \in \mathcal{S}$, we have $r^+(s) \leq r^*(al_i)$, so clearly the result of the approximate analysis is an overapproximation of the precise result.

The soundness of the approximate analysis can be decomposed into two steps:

- For any approximate scenario al_i, let $dom(al_i)$ denote the set of precise scenarios whose i-th component is equal to al_i. We say that al_i *dominates*

all precise scenarios in $dom(al_i)$. Clearly,

$$\max_{s \in dom(al_i)} r^+(s) \leq r^*(al_i) \tag{25}$$

- The set Al_i of approximate scenarios dominates all precise scenarios in $\mathcal{S}$.

The two-level combined analysis is based on the following observation: If the WCRT obtained for an approximate scenario al_i 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 al_i.

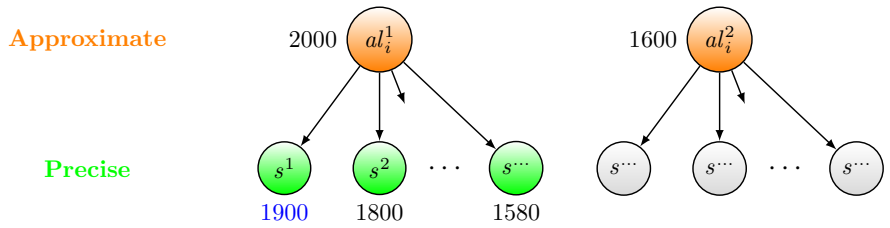

Figure 6: Scenario domination according to the two-level combined analysis for a system of four transactions. Each approximate scenario (orange node) dominates 8000 precise scenarios (green nodes). Unprinted nodes and the nodes in gray represent the scenarios which do not need to be analyzed.

Example 1. *Consider a system of four transactions* $\{T_1, T_2, T_3, T_4\}$ *where the number of alignments in each transaction is 2, 20, 20, 20 respectively. For a task of transaction* T_1*, the precise analysis has to perform 16000 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 8000 precise scenarios is depicted in Figure 6, 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* $[(2000, al_i^1); (1600, al_i^2)]$*. 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* 8000 *precise scenarios instead of* 16000*.*

The two-level combined RTA returns the same WCRT as the precise analysis but uses far 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 analyze 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. 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 a generic analysis parameterized by $\mathcal{T} = \{T_1\}$, which corresponds to the approximate analysis. The set of scenarios is thus Al_1. Then, we refine the result by adding one transaction to $\mathcal{T}$, for example T_2. Thus, we must now check more scenarios, namely all scenarios in $Al_1 \times Al_2$. Consequently, the time complexity increases but the result is more precise. If needed, all transactions can be moved into $\mathcal{T}$ such that the analysis ends up checking all precise scenarios. 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 parameterization 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 7). The full combined analysis first computes the sorted list* $[(2000, al_1^1); (1600, al_1^2)]$ *representing approximate scenarios (called level-1 scenarios) with* $\mathcal{T} := \{T_1\}$*. As a second step, we move one transaction to* $\mathcal{T}$ *and compute the results for the 10 scenarios at level 2 that are dominated by the scenario corresponding to the largest approximate response time (*2000*) at level 1, namely* al_1^1*. The results for the 10 scenarios are then sorted in descending order. That builds a part of Level-2. Recursively, we add one more transaction to* $\mathcal{T}$ *to compute level-3, and so on until level-4 when all transactions have been moved to* $\mathcal{T}$*. 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 times 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 thus returns 1900 *which is the precise WCRT. It only had to consider* 42 *approximate and* 20 *precise scenarios.*

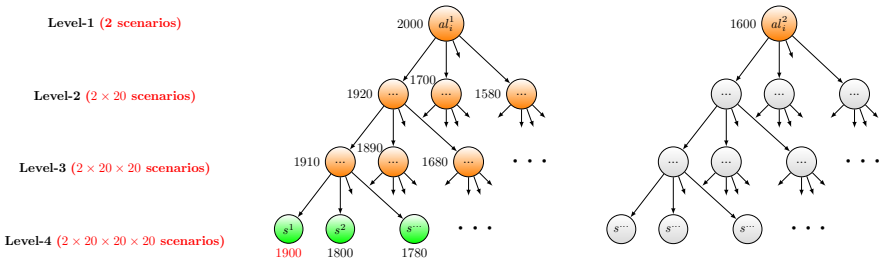

Figure 7: Scenario domination according to the full combined analysis for a system of four transactions. Each approximate scenario dominates 20 less approximate scenarios. Unprinted nodes and the nodes in gray represent the scenarios which do not need to be analyzed.

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. 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.

The structure of the full combined analysis is shown in Algorithm 1. First, the generic analysis is applied to each scenario at the first level (that is, every al_i in Al_i). The results (*i.e.,* one WCRT for each scenario) paired with their corresponding scenario are sorted in descending order, as shown at Line 3 in Algorithm 1: $l_S := sort\,(map\,(\lambda s.(r^\star(s), s))\ Al_i)$. 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 the given 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 of scenarios to analyze, and a list l_{tr} of transactions to be refined (*i.e., Sys*$\backslash\mathcal{T}$). CRTA considers each approximate WCRT of list l_s in turn, starting with the head (r, s) of l_S — remember that, by construction of l_s, $r = r^\star(s)$ is the WCRT obtained using the generic analysis on scenario s. If $R \geq r$ then it stops and returns R (this 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

Algorithm 1 Full Combined RTA

```
1: % R denotes the current WCRT, its initial value is 0
2: % l_s is initialized with the list of approximate scenarios paired with their
   corresponding WCRT
3: l_s := sort(map(λs.(r*(s), s)) Al_i)
4: % l_tr denotes the list of transactions to be refined (i.e., Sys\T)
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.(r*(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 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 refinement mechanism and sorts their results in 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 verify the results of CAN analysis tools. Consider a result R_{in} computed

by an industrial analyzer. To verify this result, we apply the full combined analysis with an initial WCRT set to R_{in}.

Algorithm 2 The CertiCAN Result Certifier

```
1: % R_in contains the WCRT to certify
2: % l_s is initialized with the list of approximate scenarios paired with their
   corresponding WCRT
3: l_s := sort(map(λs.(r*(s), s)) Al_i)
4: % l_tr denotes the list of transactions to be refined (i.e., Sys\T)
5: procedure CertiCAN(l_s, l_tr)
6:     match l_s with
7:     % if l_s is empty, returns True
8:     | nil ⇒ return True
9:     % otherwise, takes one element of l_S to analyze
10:    | (r, s) :: l'_S ⇒
11:    % if the WCRT to certify is greater than the approximate result r,
12:    then returns True
13:    if R_in ≥ r then return True
14:    else
15:        match l_tr with
16:        % if l_tr is empty, returns False
17:        | nil ⇒ False
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.(r*(s), s)) (refine s l_tr)));
23:        % certifies the result for those dominated scenarios
24:          R_local ← CertiCAN(l_local, l'_tr);
25:        if R_local = False then return False
26:        else
27:            % recursively, certifies the results for the remaining elements l'_s
28:            CertiCAN(l'_s, l_tr)
29:        end if
30:    end if
31: end procedure
32: CertiCAN(l_s, l_tr)
```

The algorithm of CertiCAN is shown in Algorithm 2. To check that R_{in} 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_{in} then the verification 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_{in}. 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_{in} 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 verification 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_{in} is greater than or equal to the precise WCRT.

5 Optimizations

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. We determine a domination relation between alignments such that we can 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.

5.1 Removing dominated alignments

Each alignment of a transaction corresponds to a workload function. For instance, for the alignment represented by the red dashed line (at time 2) in Figure 4, its cumulated workload for a given duration Δ is $wl_1(\Delta)$ in Figure 5. 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* $\Delta \in [0, L]$,

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

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 *HP*, *i.e.,* the 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,*

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

Lemma 22. *Consider two scenarios* s_1 *and* s_2*. If* wl^{s_1} *weakly dominates* wl^{s_2} w.r.t. *HP, then* $r^\star(s_1) \geq r^\star(s_2)$.

As a result, we can look for weak domination relations between workload functions, then filter out dominated functions while preserving the soundness of the analysis. Our 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

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
```

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, it examines the remaining functions l'_2 without changing l_1 (Line 11).

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

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 verify a system we need to analyze all its tasks. Workload functions will be evaluated numerous

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)
```

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 the results of workload functions.

For each transaction, we can pre-calculate each workload function up to a large enough value,[6] *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 5 when their workload values change. For instance, in Figure 5, the instants when wl_1 must be computed are 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 analyzed first? More generally, which order of transactions should be chosen by the refinement?

[6] 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.

We investigated whether the order in which transactions are considered significantly affects 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 the 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 formally proved tools: a precise analyzer,[7] 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 tools in terms of performance and scalability. These experiments can be reproduced following the instructions in CertiCAN (2021).

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

[7] 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.

The evaluated task sets are generated by NETCARBENCH,[8] 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 the set of parameters presented in Table 1. More detail about configuration parameters[9] 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.

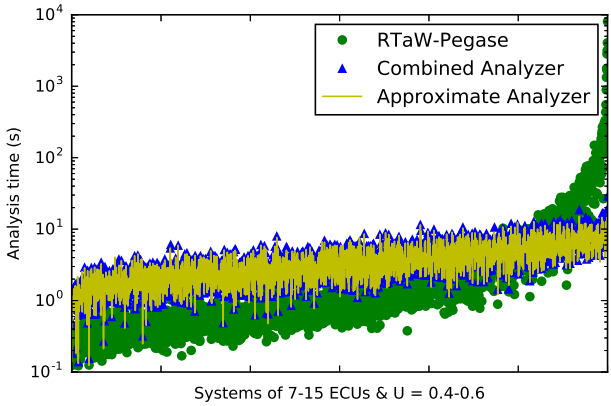

Figure 8: Comparison between verified analyzers and RTaW-Pegase

Figure 8 shows that the combined analyzer has remarkable performance. It returns precise results and its time efficiency is close to that of the approximate analyzer. Compared to RTaW-Pegase, the combined analyzer has a better scalability. For the most complex systems, RTaW-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

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

[9] 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%.

other hand, RTaW-Pegase is faster than the combined analyzer on simple systems. One reason is that RTaW-Pegase is written in C whereas the analyzer is extracted from Coq proofs in OCaml.

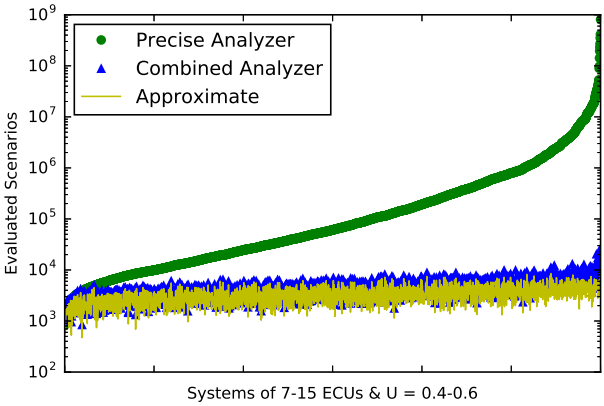

Figure 9: Evaluated scenarios by the verified analyzers

During our experimentations, we recorded the number of scenarios evaluated by the verified tools as shown in Figure 9. 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 8 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.

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

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 faster 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 different time complexities, *e.g.,* an approximate scenario requires more computations to find the maximum workload among all its workload functions.

Of course, CertiCAN is much 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 faster 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 filtering out dominated functions make it possible. The filter with SSL is 5 - 8 times faster than the one with LCM. Avoiding recomputations[10] 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 (Monot et al, 2012; Quinton et al, 2014). This indicates that CertiCAN can provide formal guarantees for industrial CAN bus analyzers.

It came as a surprise to discover that our analyzer was faster than RTaW-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 RTaW-Pegase. Integrating these optimizations into RTaW-Pegase would make it faster than our verified tools, thus allowing it to analyze even larger systems.

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

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 result certifier of Section 4 and their optimizations of Section 7. Our proofs build upon the Prosa library and use the basic definitions that 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 in CertiCAN (2021).

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

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

much more time and effort than on 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. Note that, in this paper, as in our Coq formalization, we have only considered Tindell's model of transactions with offsets. Mixed task models including periodic and sporadic tasks and/or periodic bursts can easily be taken into account by instantiating our workload functions differently.

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 CertiCAN 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. Prosa is the largest effort to date regarding the formal verification of real-time systems analyses. It is however not the first one. Previous publications in the area include Dutertre (2000), Dutertre and Stavridou (2000) and Ha et al (2004), 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 last on the scheduler of the DEOS real-time operating system. While related to our work in a broader sense, these contributions do not tackle the problem of verifying RTA results.

Closer to us, a recent attempt by Mabille et al (2013) aims at verifying the results of Network Calculus computations using the interactive proof assistant Isabelle/HOL. In particular, Mabille et al (2013) makes a case for result verification. 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 Mabille et al (2013).

Our combined RTA follows a principle that is similar to the abstraction refinement method used in Stigge et al (2014) and Stigge and Yi (2015). In particular, these two papers already use two different abstraction levels to compute precise bounds with increased efficiency. The main difference is that Stigge et al (2014) 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.

Regarding CAN analyses, the original RTA for the CAN protocol was presented by Tindell and Burns (1994) and Tindell et al (1994, 1995), then was revised and corrected by Davis et al (2007). In order to capture more CAN communication characteristics and to perform more precise analyses, the RTA of CAN has been extended by many : Du and Xu (2009); Yomsi et al (2012) considered offsets, Mubeen et al (2011, 2012, 2013) analyzed mixed messages with offsets, Davis et al (2011); Davis and Navet (2012); Davis et al (2013) presented RTAs of CAN with FIFO queues, etc. Regarding CAN analyzers,

besides RTaW-Pegase, there exist many other tools such as Volcano Network Architect by Mentor Graphics (2006), Rubus-ICE by Arcticus Systems AB (2022), CANalyzer by Vector (2022), SymTAS (2004) by Luxoft and MPS-CAN by Mubeen et al (2014). Note that CertiCAN can certify the results produced by any CAN analyzer as long as it considers the same system model.

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 the analyses and the 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 verified tools.

We have shown that the CertiCAN approach, which is based on result verification rather than tool verification, 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 verification 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.

Acknowledgments

This work has been partially supported by the French national research organization ANR (grant ANR-17-CE25-0016) through the RT-PROOFS project. We would like to thank RealTime-at-Work for granting us an academic license for RTaW-Pegase and Jiajie Wang for providing an XML parser.

References

Arcticus Systems AB (2022) Rubus ICE: Integrated Component Model Development Environment. https://www.arcticus-systems.com/products/rubus-tool-suite/

Bletsas K, Audsley NC, Huang W, et al (2018) Errata for three papers (2004-05) on fixed-priority scheduling with self-suspensions. LITES 5(1):02:1–02:20

Bosch (1991) CAN specification version 2.0

Cerqueira F, Stutz F, Brandenburg BB (2016) Prosa: A case for readable mechanized schedulability analysis. In: Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on, IEEE, pp 273–284

CertiCAN (2021) A Coq tool to certify CAN analyses and their results. https://team.inria.fr/spades/certican2/

CompCert (2021) The CompCert C compiler. https://www.absint.com/compcert/

Coq (2021) The Coq proof assistant. http://coq.inria.fr

Davis RI, Navet N (2012) Controller area network (can) schedulability analysis for messages with arbitrary deadlines in fifo and work-conserving queues. In: 2012 9th IEEE International Workshop on Factory Communication Systems, IEEE, pp 33–42

Davis RI, Burns A, Bril RJ, et al (2007) Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems 35(3):239–272

Davis RI, Kollmann S, Pollex V, et al (2011) Controller area network (can) schedulability analysis with fifo queues. In: 2011 23rd Euromicro Conference on Real-Time Systems, IEEE, pp 45–56

Davis RI, Kollmann S, Pollex V, et al (2013) Schedulability analysis for controller area network (can) with fifo queues priority queues and gateways. Real-Time Systems 49(1):73–116

Du L, Xu G (2009) Worst case response time analysis for can messages with offsets. In: 2009 IEEE International Conference on Vehicular Electronics and Safety (ICVES), IEEE, pp 41–45

Dutertre B (2000) Formal analysis of the priority ceiling protocol. In: 21st IEEE Real-Time Systems Symposium (RTSS), pp 151–160

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

Fradet P, Guo X, Monin JF, et al (2018) A generalized digraph model for expressing dependencies. In: RTNS'18-26th International Conference on Real-Time Networks and Systems, pp 1–11

Fradet P, Guo X, Monin JF, et al (2019) 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, pp 182–191

Guo X, Quinton S, Fradet P, et al (2017) Work-in-progress: Toward a Coq-certified tool for the schedulability analysis of tasks with offsets. In: Real-Time Systems Symposium (RTSS), 2017 IEEE, IEEE, pp 387–389

Ha V, Rangarajan M, Cofer D, et al (2004) Feature-based decomposition of inductive proofs applied to real-time avionics software: An experience report. In: 26th Conference on Software Engineering (ICSE), pp 304–313

Isabelle (2021) The Isabelle proof assistant. https://isabelle.in.tum.de/

Leroy X (2009) Formal verification of a realistic compiler. Communications of the ACM 52(7):107–115. URL http://xavierleroy.org/publi/compcert-CACM.pdf

Mabille E, Boyer M, Fejoz L, et al (2013) Towards certifying network calculus. In: Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings, pp 484–489

Mentor Graphics (2006) Volcano Network Architect. https://theeshadow.com/files/volvo/can/VNA_Datasheet.pdf

Monot A, Navet N, Bavoux B, et al (2012) Fine-grained simulation in the design of automotive communication systems. In: ERTSS-Embedded Real Time Software and Systems-2012

Mubeen S, Mäm-Turja J, Sjödin M (2011) Extending schedulability analysis of controller area network (can) for mixed (periodic/sporadic) messages. In: ETFA2011, IEEE, pp 1–10

Mubeen S, Mäki-Turja J, Sjödin M (2012) Worst-case response-time analysis for mixed messages with offsets in controller area network. In: Proceedings of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012), IEEE, pp 1–10

Mubeen S, Mäki-Turja J, Sjödin M (2013) Extending offset-based response-time analysis for mixed messages in controller area network. In: 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, pp 1–10

Mubeen S, Mäki-Turja J, Sjödin M (2014) Mps-can analyzer: Integrated implementation of response-time analyses for controller area network. Journal of Systems architecture 60(10):828–841

Nipkow T, Paulson LC, Wenzel M (2002) Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol 2283. Springer,

URL https://doi.org/10.1007/3-540-45949-9

Prosa (2017) A Library for formally proven schedulability analysis. http://prosa.mpi-sws.org/

Quinton S, Bone TT, Hennig J, et al (2014) 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, pp 44:1–44:6

RTaW-Pegase (2021) RTaW-Pegase: a Tool for Modeling, Simulation and automated Configuration of communication networks. http://www.realtimeatwork.com/software/rtaw-pegase/

Sel4 (2018) The seL4 microkernel. https://sel4.systems/

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

Stigge M, Guan N, Yi W (2014) Refinement-based exact response-time analysis. In: 26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014, pp 143–152

SymTAS (2004) SymTA/S: Model-based timing analysis and optimization. https://auto.luxoft.com/uth/timing-analysis-tools/

Tindell K (1992) Using offset information to analyse static priority preemptively scheduled task sets. Technical report YCS 182, University of York, Department of Computer Science

Tindell K (1994) Adding time-offsets to schedulability analysis. University of York, Department of Computer Science

Tindell K, Burns A (1994) Guaranteeing message latencies on controller area network (CAN). In: Proc. of 1st international CAN conference, pp 1–11

Tindell K, Hanssmon H, Wellings AJ (1994) 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, pp 259–263

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

Vector (2022) Analyzing ECUs and Networks with CANalyzer. https://www.vector.com/int/en/products/products-a-z/software/canalyzer/#

Yomsi PM, Bertrand D, Navet N, et al (2012) Controller area network (CAN): Response time analysis with offsets. In: 2012 9th IEEE International Workshop on Factory Communication Systems, pp 43–52

Appendix 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

Appendix 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