Latency analysis for data chains of real-time periodic tasks

Tomasz Kloda, Antoine Bertout, Yves Sorel

Latency analysis for data chains of real-time periodic tasks

Tomasz Kloda
University of Modena and Reggio Emilia
Modena, Italy
tomasz.kloda@unimore.it

Antoine Bertout
Université de Poitiers
Poitiers, France
antoine.bertout@univ-poitiers.fr

Yves Sorel
Inria
Paris, France
yves.sorel@inria.fr

***Abstract*—A data chain is a sequence of periodic real-time communicating tasks that are processing the data from sensors up to actuators. It determines an order in which the tasks propagate data but not in which they are executed: inter-task communication and scheduling are independent. In this paper, we focus on the latency computation, considered as the time elapsed from getting the data from an input and processing it to an output of a data chain. We propose a method for the worst-case latency calculation of periodic tasks' data chains executed by a partitioned fixed-priority preemptive scheduler upon a multiprocessor platform. As far as we know, there is no such formal approach based on closed-form expression for communicating real-time tasks.**

I. Introduction

An embedded real-time system must react to its surroundings by sensing, computational processing and actuating upon external events within strict timing constraints. In large and complex industrial systems, different tasks process in a specific order the data before the final results can be delivered to the actuators. From a sensor to an actuator, a specific chain of tasks is involved in the computational processing.

Task chains. A *task chain* is a sequence of communicating tasks in which every task receives the data from its predecessors [1]. According to the activation rule, two models are commonly distinguished: trigger and data chains. Tasks in the *trigger* chains are activated by the events issued from the preceding tasks. In the *data* chains, the tasks are independent and are activated recurrently at their individual rates.

Trigger chains are characterized by a more flexible timing behavior. Tasks of the same chain execute only if necessary: a task is scheduled only if it receives the request from its predecessor. It results in a more efficient use of the processor resources and a more reactive inter-task data passing. To impose the precedence between subsequent tasks, trigger chains require task synchronization (e.g., semaphores). The use of the synchronization primitives can lead to priority-inversion problems and possible deadlock formations [2], [3]. An additional effort is needed for the integration of these primitives together with adequate control structures. On top of that, as safety-critical software is subject to stringent safety standards, adding these features to the system makes the certification process more burdensome [4].

Data chains are not synchronized and follow a strict activation pattern. Tasks are released at every period even if there is no new data to consume. Certain delay of time can elapse from the instant when the precedent task produces the data and the instant when the current task is released and can consume it. However, especially in large industrial applications, data chains may be preferred over trigger chains. Their unrestrained execution model can permit to reduce time and cost overheads related to the implementation and validation of synchronization mechanisms needed for trigger chains.

Task chain latency. Real-time systems comprised of task chains are subject to different timing requirements. Wyss et al. [5] summarize them as follows. *Reactivity* (or data reaction [1]) is a time interval during which a data must be present at the sensor input in order to be detected. *Latency* is a time interval elapsed between arrival of data at the sensor input and actuator's update with value corresponding to that particular data (i.e., time needed for data to be propagated from sensor to actuator). *Freshness* (or data age [6]) is a time interval during which the actuator's value corresponding to the same sensor data instance is in use.

Contribution. In this work, we focus on the computation of the worst-case latency for data chains. Given a data chain of periodic independent tasks with implicit deadlines scheduled by a fixed-priority preemptive partitioned algorithm upon identical multiprocessor platforms, a priority assignment of these tasks and the worst-case response times of their jobs, we propose a method for the computation of the data chain worst-case latency. We sketch simple algorithms for an easy and efficient implementation of the proposed method on top of any response time analysis tool.

Paper structure. The paper is organized as follows. In the next section we review related work. The data chain model is defined and hypotheses are made in Section III. A method for the calculation of the worst-case latency of data chains is presented in Section IV. In Section V we evaluate our method using the generic task sets based on a real-world automotive system and in Section VI we conclude the paper.

II. Related work

Davare et al. [7] propose period and priority assignment technique for the periodic tasks running on different nodes without consistent view of time (no clock synchronization).

Gerber et al. [8], [9] consider a task chain with harmonicity constraint and enforce by the offset and priority selection that the producer task executes always before the consumer task. Di Natale et al. [10] and Davare et al. [7] specify the general case when the producer and consumer have harmonic periods without being necessarily monotonic in their lengths over the entire chain. They assume additionally that the relative offsets are selected to enforce the execution of the producer before the consumer.

Feiertag et al. [6] and Mubeen et al. [1] developed frameworks for the computation of different latencies under the same model as in the present work upon a single processor. The framework presented in [1] is a part of Rubus-ICE tool suite used for the development of automotive systems and handles also other message passing abstractions typical for this kind of systems (e.g., CAN messages). Both frameworks compute all the reachable paths while in our work we propose a closed-form expression (Lemma 1) that identifies directly the one that gives a maximum latency. Rajeev et al. [11] and Anwikar and Badhuri [12] use model-checking based techniques to analyze latencies in the distributed systems in which chains are composed of preemptive tasks and non-preemptive messages. Khatib et al. [13] use Synchronous Data-flow Graph to model the communications between multi-periodic tasks that start exactly at their releases.

Wyss et al. [5], [14] present an analysis of the end-to-end latencies for a formal language with synchronous semantics Prelude [15]. The data propagation in functional chain is expressed by a data dependency word [16].

Becker et al. [17]–[19] analyze the end-to-end latency for periodic tasks in abstraction from a concrete scheduling policy, for the static off-line schedules [19], [20] and also using the knowledge of the worst-case response times of the tasks [20]. The framework presented in [20] is versatile and can be applied for the most timing analysis in automotive and avionic systems on single processor like tasks with offsets and Logical Execution Time model; it also automatically generates the job-level dependencies to meet latency constraints. The method is based on the tree of all possible data propagation paths. For two communicating tasks, their communication time instant leading to maximum latency can be identified once all their previous communication time instants are generated. The approach proposed in our work uses the scheduling information (priority, response time, processor assignment) to identify directly with a closed-form expression (Equation (4)) the communication time instant leading to the maximum latency. The focus of our work is on multiprocessor platforms and considers that response time of a particular job is also a function of its release time. We formally validate our method by analyzing different schedules resulting from the non-constant execution times of the tasks.

The holistic analysis [21] applies to the tasks whose execution can be triggered by the arrival of a message or by the completion of a preceding task. Schlatow and Ernst [3] proposed an upper bound on the end-to-end latency of task chains implementing synchronous and asynchronous communicating threads. The multiframe model [22] can be used to express the sequential execution of communicating jobs. It generates cyclically a fixed sequence of different jobs according to a predefined time pattern.

III. System model

We introduce a data chain as a sequence of independent periodic tasks with implicit deadlines scheduled by a partitioned fixed-priority preemptive policy upon a multiprocessor platform.

A. Periodic task model

A system consists of a finite set of periodic tasks. Each task τ_i is characterized by its *worst-case execution time* C_i and its *period* T_i (both positive integers). An instance (job) of task τ_i is released every period T_i and must meet its execution requirement upper bounded by C_i by the arrival of its next instance (implicit deadline $D_i = T_i$). The first instance of τ_i is released at the time instant 0. The infinite set of all the release time instants of task τ_i is referred to as $A(\tau_i) = \{0, T_i, 2T_i, \ldots\}$. An instance of the task τ_i released at $r_{i.j} \in A(\tau_i)$ is denoted by $J_i(r_{i.j})$. In the rest of the paper, to facilitate the reading, an arrival $r_{i.j}$ of a task τ_i will be sometimes simply denoted by r_i when there is no possible ambiguity with different arrivals of τ_i. The instances of task τ_i are executed upon the processor $P(\tau_i)$ (partitioned scheduling). We suppose that the task-to-processor assignment is already given. All the processors are identical and have the consistent view of time (perfectly synchronized clocks). The *hyperperiod* H is the least common multiple of all the tasks periods.

Tasks are scheduled upon every processor by a fixed-priority preemptive scheduler. Each task τ_i is assumed to have a unique priority $\pi(\tau_i)$. We consider that τ_i has a higher priority than τ_k if $\pi(\tau_i) > \pi(\tau_k)$. Additionally, we denote a set of tasks with priorities higher than $\pi(\tau_i)$ that execute on $P(\tau_i)$ as: $hp(\tau_i) = \{\tau_k : \pi(\tau_k) > \pi(\tau_i)\}$ and with priorities higher than or equal to $\pi(\tau_i)$ as: $hep(\tau_i) = hp(\tau_i) \cup \{\tau_i\}$. The tasks are assumed to be independent and their executions cannot be blocked by another task other than due to contention on processor. Once a task starts to execute it will not voluntarily suspend its execution. Tasks can begin processing, complete or be preempted at any time with respect to their parameters.

B. Task response time

The worst-case *job response time* $R_i(r_{i.j})$ of task τ_i released at $r_{i.j} \in A(\tau_i)$ is given by the longest time from $r_{i.j}$ until it completes its execution. The time instant

when it completes its execution is denoted by $f_i(r_{i,j}) = r_{i,j} + R_i(r_{i,j})$.

The worst-case *task response time* of τ_i, denoted as R_i, is given by the maximum worst-case job response time over all its jobs: $R_i = \max_{r_{i,j} \in A(\tau_i)} R_i(r_{i,j})$. A system is deemed schedulable when $\forall \tau_i : R_i \leq T_i$.

C. Data chain model

A data chain F_n is a sequence of $n \geq 1$ tasks $(\tau_1, \ldots, \tau_n)$ describing a flow of communication between the tasks realized via shared registers. Let $head(F_n) = \tau_1$ be its first task, $last(F_n) = \tau_n$ its last task and $tail(F_n) = (\tau_2, \ldots, \tau_n)$ the chain obtained by removing τ_1. If $n = 1$, then τ_1 after its start reads the data from its sensor, computes its outputs and writes them to the appropriate actuator. Otherwise, if $n \geq 2$, the data chain F_n processes the data as follows:

- when τ_1 starts, it reads the data from a sensor, computes the results and, at the end of its execution, writes these results into the register of τ_2;
- when τ_i (for $i : 1 < i < n$) starts, it reads the data from the register of τ_{i-1}, computes the results and writes them, at the end of τ_i, into the register of τ_{i+1};
- when τ_n starts, it reads the data from τ_{n-1}, computes the outputs and, at the end of its execution, writes these outputs to the corresponding actuator.

The start of the tasks does not depend on the communication, tasks are scheduled according to the assigned priorities only. The same mechanism is also implemented in Cyclical Asynchronous Buffers [23]. Figure 1 illustrates a data chain F_3 of three tasks (τ_1, τ_2 and τ_3) together with their registers.

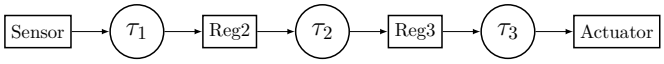

Figure 1: Register-based communication in a data chain of three tasks τ_1, τ_2 and τ_3.

The communication described above assumes the read-execute-write task semantics (implicit communication [19], [24] in AUTOSAR): the input data can be read only at the beginning of the execution and the results can be written only at its end. The operations on the registers (read and write) are atomic and are assumed to take a negligible amount of time. The computation process can be preempted at any time and resumed later from the same context. Each task has its individual register which stores only the most recent data. The registers are asynchronous in the sense that they can be accessed anytime.

D. Data propagation path

Based on the concept of timed path introduced in [6] and in [17], we define, for a data chain F_n released at time instant $r_1 \in A(\tau_1)$ such that $\tau_1 \in head(F_n)$, a set of *data propagation paths* $\Omega(F_n, r_1)$. A data propagation path $p \in \Omega(F_n, r_1)$ is a n-tuple $(r_1, \ldots, r_n)$ where $r_i \in p$ is the release time of the task τ_i instance that propagates the data. A task τ_i propagates a data when it writes this data for the first time into a task register of the next task τ_{i+1}. It is supposed that at time instant r_1 all the task registers are empty and the data is from that moment continuously present at the chain's input.

Figure 2 illustrates two different scenarios of execution of a data chain F_3. The chain is composed of three tasks: τ_1, τ_2 and τ_3 such that $\pi(\tau_2) > \pi(\tau_3) > \pi(\tau_1)$. In this figure, two data propagation paths can be distinguished for $\Omega(F_3, r_{1,1})$. The first one, marked with the dotted line, when τ_1 terminates at $f'_{1,1} : (r_{1,1}, r_{2,2}, r_{3,2})$. The second one, marked with the solid line, when τ_1 terminates later at $f_{1,1} : (r_{1,1}, r_{2,3}, r_{3,3})$ with $f_{1,1} > f'_{1,1}$.

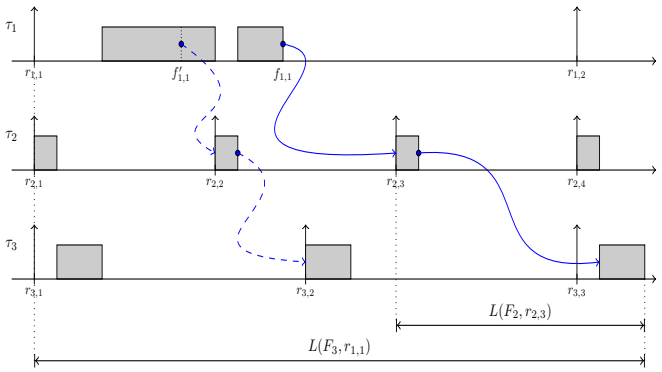

Figure 2: Two execution scenarios of a data chain of three tasks: τ_1, τ_2 and τ_3.

E. Data chain latency

Latency of a data chain is a delay between the arrival of its input and the first output corresponding to it [11].

Definition 1 (Latency)**.** Let F_n be a data chain whose registers are initially empty. Its data latency is a time interval elapsed between

- the time instant at which the data arrives at the input sensor connected to $head(F_n)$ given that the data is available sufficiently long to be detected, and
- the time instant at which the actuator connected to $last(F_n)$ is updated with the data corresponding to the computation performed by all the tasks of F_n.

Given that the data arrives at r_1, the maximum latency $L(F_n, r_1)$ of a data chain F_n corresponds to the data propagation path in which the last task completes at the latest time.

Definition 2 (Maximum data chain latency at r_1)**.** Let F_n be a data chain of $n \geq 1$ tasks and $r_1 \in A(\tau_1)$ the release time of its first task $\tau_1 = head(F_n)$, and the input of τ_1 arrives in its register at r_1. The maximum latency of data chain F_n released at r_1 is not greater than:

$$L(F_n, r_1) = \max_{p \in \Omega(F_n, r_1)} f_n(r_n) - r_1 \qquad (1)$$

where $r_n \in p$ and $\tau_n = last(F_n)$.

IV. Latency Analysis

We propose a method for computing the maximum latency of the data chain if the input from the sensor arrives at the release of its first task and then we generalize this method for the arbitrary input arrival times to obtain the worst-case latency. First, for a task that writes the data and the next task in the chain that subsequently reads this data, we characterize the time distances between their releases. Given a data arrival time, we construct a sequence of releases of the tasks propagating this data from the first to the last task in the chain (data propagation path). Then, to obtain the worst-case latency, we expand the proposed method over all data arrival time instants.

A. The latest data propagation time

We characterize the largest delay in data passing for a pair of directly communicating tasks. Given a release time of the task propagating data (producer), we find the latest possible release time instant of the task that reads this particular data for the first time (consumer).

In the example shown on Figure 2, there are two instances of task τ_2 that can propagate the data. The solid line represents the inter-task data passing when τ_1 finishes at $f_{1,1}$ (suppose that $f_1(r_{1,1}) = f_{1,1}$) and the data is consumed by τ_2 released at $r_{2,3}$. The dashed line corresponds to the case when τ_1 finishes at $f'_{1,1}$ ($f'_{1,1} < f_{1,1}$) and the data is consumed by τ_2 released at $r_{2,2}$ ($r_{2,2} < r_{2,3}$). Task τ_2 instance that consumes the data propagates it further to τ_3. In this example, the data from τ_1 can be propagated to τ_3 by the instances of τ_2 released at $r_{2,2}$ or at $r_{2,3}$. The time instant $r_{2,3}$ is the latest possible release of τ_2 that can propagate the data.

Let F_2 be a data chain of two communicating tasks: producer τ_p and consumer τ_c having respectively periods T_p and T_c. The producer, at the end of its execution, fills the input register of the consumer with the result of its computation. Let r_c be the latest release time of the consumer job that has for the first time in its register the data written by the producer released at r_p.

First, suppose that the consumer has higher priority than producer: $\pi(\tau_c) > \pi(\tau_p)$ and is mapped to the same processor: $P(\tau_c) = P(\tau_p)$. Figure 3 illustrates a sample execution of such two tasks. Due to the priority order, whenever consumer τ_c is released it can preempt

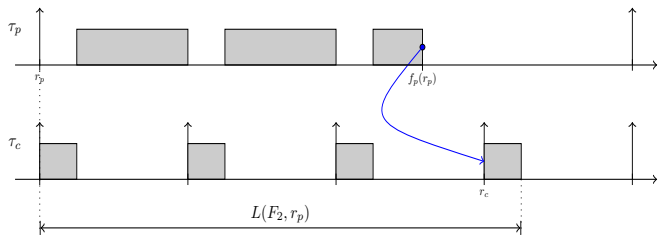

Figure 3: Communicating tasks: consumer τ_c has higher priority.

producer τ_p. Consumer checks for the data at each release but the data cannot be written into the register before the end of the producer execution. Therefore, the data is read for the first time by the consumer instance released immediately after the end of the producer. At worst, the producer finishes at $f_p(r_p)$ and the data is read by the earliest consumer instance released after $f_p(r_p)$:

$$r_c = \left\lceil \frac{f_p(r_p)}{T_c} \right\rceil T_c \tag{2}$$

Now, suppose that producer has higher priority than consumer: $\pi(\tau_p) > \pi(\tau_c)$ and is still mapped to the same processor: $P(\tau_c) = P(\tau_p)$. Suppose also that there is a third task τ_h mapped to the same processor which is not included in the chain. Task τ_h has the highest priority, $\pi(\tau_h) > \pi(\tau_p) > \pi(\tau_c)$, and its period is equal to the period of consumer, $T_h = T_c$. A sample run of such three tasks is shown in Figure 4. Consumer has lower priority

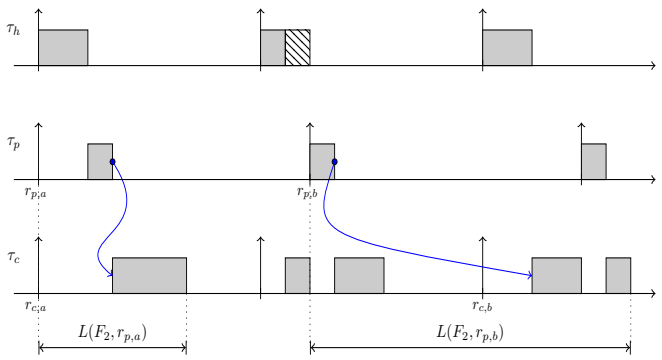

Figure 4: Communicating tasks: producer τ_p has higher priority.

and cannot preempt producer. Suppose that producer and consumer are:

a) released at the same time (see time instant $r_{p,a}$ in Figure 4): Due to the priority order, consumer cannot start until producer completes its execution. When it completes, it writes the data into register. Consumer can read the data at the beginning of its execution.

b) not released at the same time (see time instant $r_{p,b}$ in Figure 4): The consumer instance released immediately before the release of producer may be blocked by the higher priority jobs until the release of producer. In that case, it starts after the end of producer execution and reads the data. Nonetheless, the higher priority jobs may complete earlier than their worst-case execution times (like the second τ_h instance in Figure 4 where this scenario is illustrated by the hatched part) or their worst-case execution times may be too short to block consumer until the release of producer. The consumer instance that starts before the release of producer cannot get the data from this particular instance of producer. It is the first consumer instance released after the release of producer that reads the data.

At worst, the data is read for the first time by the first consumer instance released at or after the release of producer:

$$r_c = \left\lceil \frac{r_p}{T_c} \right\rceil T_c \tag{3}$$

We generalize the results obtained above for a chain of tasks allocated to different processors. We consider that the producer τ_p and the consumer τ_c execute on two different processors with perfectly synchronized clocks: $P(\tau_c) \neq P(\tau_p)$. Since the tasks cannot block each other, we can assume that every consumer instance starts always as soon as possible and its buffer is empty if the producer is still running on the other processor. At worst, the data is read for the first time by the consumer instance released immediately after the end of the producer (see Equation (2)).

Lemma 1. In a schedulable task set with two communicating tasks τ_p and τ_c such that τ_p writes the results of its computation to the register of τ_c, the latest possible release r_c of the instance of τ_c reading for the first time the data propagated by the instance of τ_p released at $r_p \in A(\tau_p)$ is given by:

$$r_c = \begin{cases} \left\lceil \frac{f_p(r_p)}{T_c} \right\rceil T_c & \text{if } \pi(\tau_p) < \pi(\tau_c) \text{ or } P(\tau_p) \neq P(\tau_c) \\ \left\lceil \frac{r_p}{T_c} \right\rceil T_c & \text{if } \pi(\tau_p) > \pi(\tau_c) \text{ and } P(\tau_p) = P(\tau_c) \end{cases} \tag{4}$$

where T_p and T_c are periods of τ_p and τ_c respectively.

B. The longest data propagation path

We characterize the data propagation path that gives a maximum latency value. In such data propagation path: i) the tasks propagating the data are released as late as possible (according to their periodic behaviour), and ii) their instances can attain their worst-case job response times.

For instance, in Figure 2, the data path marked with a solid line satisfies the above mentioned conditions and the value of its latency is maximal. Propagating the data earlier, as for the data path with a dotted line, cannot increase the latency value.

If there exists a data propagation path in which the data propagation occurs for each task as late as possible and its tasks terminate as late as possible, then it leads to the maximum latency. If such path cannot be constructed in the schedule, then the latency value associated with this path cannot be lower than the actual value of the latency given by some realizable path. The following lemma establishes this basic property. Its results are used later in Theorem 1.

Lemma 2. For any data chain of $n \geq 1$ schedulable tasks, the data propagation path p in which each task instance that propagates the data is released as late as possible and each task instance can terminate at its worst-case finishing time, gives the maximum latency.

Proof. Suppose that $p = (r_1, \ldots, r_n)$ where $r_1, \ldots, r_n$ are the release time instants of the consecutive instances of tasks that propagate the data. Let $p' = (r'_1, \ldots, r'_n)$ be another path of the same chain such that $r'_1 = r_1$. First we prove by induction that $r_n \geq r'_n$. Base case ($i = 1$). By definition $r_1 \geq r'_1$. Induction step ($1 < i < n$). By the induction hypothesis: $r_i \geq r'_i$. Since the chain's tasks are schedulable and have their deadlines equal to their periods every task instance must finish before the release of its next instance. If τ_i is released before r_i at $r'_i < r_i$, then it executes within interval $[r'_i, r'_i + T_i]$ and must finish before r_i. It fills the input register of τ_{i+1} with the results of its computation before r_i. The instance of task τ_i released at r_i executes within interval $[r_i, r_i + T_i]$ and fills the register of τ_{i+1} with the results of its computation after r_i. Consequently, no instance of τ_i released at $r'_i \leq r_i$ can write into the register of τ_{i+1} at a later time than τ_i released at r_i does. Hence, if task τ_{i+1} released at r_{i+1} has already data from τ_i released at r_i then the earlier or the same instance of τ_{i+1} must have the data from τ_i released at r'_i: $r_{i+1} \geq r'_{i+1}$.

From the definition of maximum latency given by Equation (1), the lemma statement holds iff: $f_n(r_n) \geq f_n(r'_n)$. The LHS of the previous expression gives a finishing time of task τ_n released at r_n and RHS a finishing time of the same task released at $r'_n \leq r_n$. If $r_n = r'_n$, then $f_n(r_n) = f_n(r'_n)$ and the statement is clearly true. Otherwise, if $r_n > r'_n$, then by the same reasoning as in the induction part, every task instance must finish before the release of its next instance and it follows that $f_n(r_n) > f_n(r'_n)$. □

C. Recursive method for latency computation

The problem of finding the maximum latency value for a data chain of n tasks released at r_p can be divided into two sub-problems:

- finding the latest time instant r_c of data propagation from the first to the second task of the chain,
- finding the maximum latency value of a chain released at r_c that is made up of the remaining $n - 1$ tasks.

Consider again the data chain from Figure 2 with τ_1 that is released at $r_{1,1}$ and completes its execution at $f_{1,1}$. The maximum latency $L(F_3, r_{1,1})$ can be expressed as $L(F_3, r_{1,1}) = r_{2,3} + L(F_2, r_{2,3}) - r_{1,1}$ where F_2 is a data chain composed of τ_2 and τ_3.

Theorem 1. Let F_n be a data chain of $n \geq 2$ tasks and a data chain F_{n-1} of $n-1$ tasks such that F_{n-1} is obtained from F_n by removing its first task τ_p: $F_{n-1} = tail(F_n)$. The maximum latency $L(F_n, r_p)$ of F_n whose first task τ_p is released at the time instant $r_p \in A(\tau_p)$ is:

$$L(F_n, r_p) \;\geq\; (r_c - r_p) + L(F_{n-1}, r_c) \tag{5}$$

where $L(F_{n-1}, r_c)$ is the maximum latency of F_{n-1} whose first task τ_c is released at the time instant r_c given by Formula (4).

Proof. Let $(r_1, \ldots, r_n)$ be a data propagation path obtained by the iterative application of Theorem 1. For $1 < i \leq n$, Equation (4) gives r_i that is the latest possible release of τ_i instance that propagates the data. By Lemma 2, a latency value of a data propagation path with the latest possible release time instants cannot be smaller than a latency value of any other data propagation path. □

Algorithm 1 uses the results of Theorem 1 to compute the maximum latency of a data chain F_n whose first task is released at the time instant r_p. The algorithm has a linear time complexity $O(n)$ (worst-case response times are assumed to be known).

Algorithm 1 Data chain maximum latency at r_p.

Require: F_n is a data chain, $n \geq 1$, $r_p \in A(head(F_n))$

function $\mathrm{L}(F_n, r_p)$
  $\tau_p \leftarrow head(F_n)$
  if $n = 1$ **then**
    return $R_p(r_p)$
  end if
  $F_{n-1} \leftarrow tail(F_n)$, $\tau_c \leftarrow head(F_{n-1})$, $Q \leftarrow 0$
  if $\pi(\tau_p) < \pi(\tau_c)$ or $P(\tau_p) = P(\tau_c)$ **then**
    $Q \leftarrow R_p(r_p)$
  end if
  $r_c \leftarrow \left\lceil \frac{r_p + Q}{T_c} \right\rceil T_c$
  return $r_c - r_p + \mathrm{L}(F_{n-1}, r_c)$
end function

D. The worst-case latency

We derive the worst-case latency by applying the above presented analysis for all the release time instants of the first task in the chain and by taking into account the delay of the first task sensor data detection.

Since the processor may be idle at any time, a task can start to execute at the earliest at its release time. Suppose that τ_1, the first task in the chain, starts at $r_1 \in A(\tau_1)$ and the data arrives at the sensor immediately after its start. In that case, the data is handled by the next task instance released at $r_1 + T_1$. The delay experienced by the data in the sensor register is at most one task period (a similar remark is made in [7], [24]). Then, the data travels along the data propagation path as explained above. There are many paths and their trajectories depend on the release time of the first task. To find the worst-case latency value all the paths must be verified. Given a data chain F_n of $n \geq 1$ tasks, the value of its worst-case latency is:

$$L(F_n) < T_1 + \max_{r_1 \in A(\tau_1)} L(F_n, r_1) \tag{6}$$

where $\tau_1 \in head(F_n)$ and T_1 is its period.

To compute the worst-case latency, Formula (6) must be evaluated for all chain instances released at $0, T_1, 2T_1, \ldots$. To make it computationally tractable the set of considered releases must be limited. The schedule repeats cyclically every hyperperiod H (least common multiple) of all the tasks in the task set [25]. Moreover, the execution of the task is not impacted by the tasks of the lower priority and the tasks that run on different processors. It is sufficient to examine only the releases of the first chain task within interval $[0, H)$ where $H = \mathrm{lcm}\{ T_j \mid \tau_j \in hep(\tau_i), \tau_j \in F_n \}$. Depending on system properties this limit can be further reduced. If the worst-case response times of each task instance are equal, $\forall k \in \mathbb{N}, \tau_i : R_i(t) = R_i(t + kT_i)$, then it is sufficient to apply Algorithm 1 on hyperperiod of all the tasks in the data chain $H = \mathrm{lcm}\{ T_i \mid \tau_i \in F_n \}$. For instance, in the analysis of task sets with harmonic periods scheduled under *Rate Monotonic* (*RM*) all the jobs of the same task have their worst-case response times equal [26], [27]. Such hypothesis can also be made when schedulability analysis only provides a single upper bound for all task jobs.

Algorithm 2 finds a maximum latency by calling the function $L(F_n, r_1)$ defined in Algorithm 1 for all the possible releases r_1 of the first task in the chain F_n within the hyperperiod H. Then, finding the worst-case latency has a $O(n \cdot \frac{H}{T_1})$ time complexity. As the hyperperiod grows exponentially as function of the largest task period in the set and with the number of tasks, the problem of finding the worst-case latency may be intractable in some cases.

Algorithm 2 Data chain worst-case latency.

Require: F_n is a data chain of n tasks, $n \geq 1$

$H \leftarrow$ hyperperiod of all tasks in the set, $\tau_1 \leftarrow head(F_n)$
for all $r_1 \in \{0, T_1, 2T_1, \ldots, H - T_1\}$ **do**
  compute $L(F_n, r_1)$ using Algorithm 1
end for
$L(F_n) \leftarrow T_1 + \max_{r_1} L(F_n, r_1)$

Function $L(F_n, r_1)$ (Algorithm 1) when called in Algorithm 2 can be improved by saving the values of its calls in a lookup table. It may avoid recomputing the same values for the paths that merge together. In particular, if r_c is the value of Equation (4) for r_p, then the equation gives the same value for all r'_p such that: $r'_p \in [r_p, r_c]$ if $\pi(\tau_p) > \pi(\tau_c)$ and $r'_p + R_p(r'_p) \in [r_p, r_c]$ if $\pi(\tau_p) < \pi(\tau_c)$. The tasks instances released at r'_p write the data into the register of the same consumer instance and their data paths join together. All of them can follow from that point the same trajectory. The data path traversing r_p is released at the earliest time and only it can lead to the worst-case latency. All other paths going through r'_p can be omitted.

Consider Figure 5. The instances of producer task τ_p released at the time instants r_1, r_2 and r_3, such that $r_1 < r_2 < r_3$, communicate with the same instance of consumer task τ_c. The respective latencies are: $L(F_2, r_1) > L(F_2, r_2) > L(F_2, r_3)$. The latency has its greatest value for r_1 and all the subsequent releases, r_2 and r_3, as they cannot

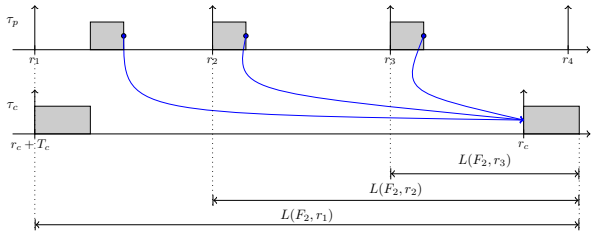

Figure 5: Non critical data propagation paths in the worst-case latency calculation.

increase this value, can be omitted in the computation of the worst-case latency.

V. Experiments

We evaluate the precision of the proposed method for the worst-case latency computation on a generic automotive benchmark [28] upon a single processor.

We compare it against a linear-time upper bound on the worst-case latency proposed by Davare et al. [7]. The method proposed by Davare et al. [7] applies in principle for the sporadic tasks but its use may be also advantageous for the periodic tasks due to its linear-time complexity. As the tasks are not coordinated, the producer τ_p may write the data into the register of consumer task τ_c immediately after its start. It induces a maximum delay of $T_c + R_c$ for each pair of producer/consumer. The worst-case latency for a chain of n tasks F_n is expressed as:

$$L(F_n) = \sum_{i=1}^{n} (T_i + R_i) \tag{7}$$

A. Chain generation

Data chains lengths vary from 2 to 20 tasks picked from a set of 50 tasks. Task parameters are generated based on the automotive benchmark [28]. Periods are randomly chosen from the set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ with an associated probability of appearance (the same as in [28]). The utilization of task τ_i is generated using the UUnifast [29] algorithm and then multiplied by the chosen period T_i giving C_i. We consider RM scheduling policy and discard unschedulable task sets. To focus on chain length only, we successively decrease the length from 20 to a minimum of 2 in such way that the number of different periods remains respected. The worst-case response times are obtained from SimSo simulator [30].

The algorithms described in this work are available on-line [1] through a Python implementation with which experiments presented below are fully reproducible.

B. Analysis precision

In the experiment, we compare the results of the worst-case latency by each method. The results of the experiment are shown in Figure 6. Each point represents the average

[1]https://www.lias-lab.fr/~antoinebertout/software/latency.zip

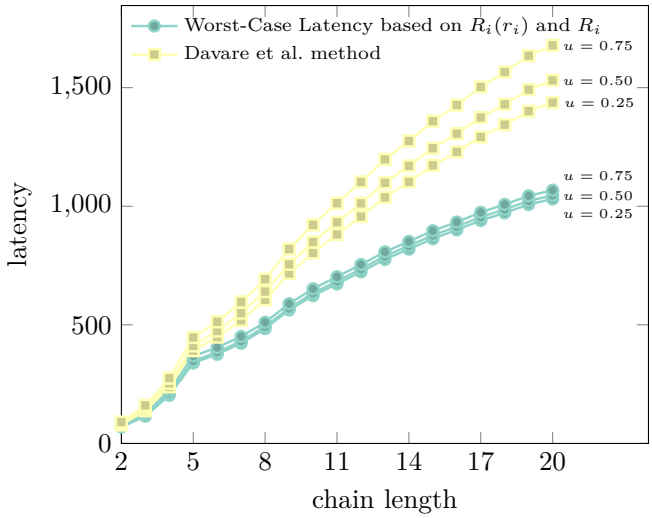

Figure 6: Impact of the different worst-case latency computation methods.

worst-case latency value of the corresponding computation method as a function of the chain length for a given utilization factor of the task set. Both methods were tested on the same set of randomly generated data chains. The results obtained are averaged on a minimum of 10000 executions for a given utilization factor $u \in \{0.25, 0.50, 0.75\}$. The number of distinct periods in the chains varies from 1 to 5. Contrary to the results given by our method, we observe that the pessimism of the linear-time upper bound increases noticeably with the length of the chain.

In this particular case, the worst-case latency calculated with the worst-case job response times is equal to the worst-case latency calculated with the worst-case task response times with a tolerance of 10^{-3}. Indeed, as task periods are quasi harmonic, most of the task instances verify $R_i(r_i) = R_i$ under RM scheduling (see also Section IV-D). Therefore, both cases are not differentiated in the figure. The method described in Algorithm 2 provides much more accurate estimates of the worst-case latency and is less sensitive to the increase of the utilization factor. The improvement is achieved with a higher computational cost. Nonetheless, the time needed to execute our method was between 2 and 6 milliseconds [2] (without a noticeable impact of the chain length) which is admissible in the most of the cases.

VI. Conclusion and future works

In this paper we proposed a method for the computation of the worst-case latency of data chains and its open-source implementation. The method is easy to implement and can be integrated independently on top of any tool that provides response time analysis.

By investigating the problem of the latency at the level of scheduling, we identify the relations between the execution

[2]Each experiment has been performed on a single Intel Xeon E5-2630 v3 CPU (20M cache, 2.40GHz).

and the communication of the real-time task and provide a solid base for the future research. In particular, Lemma 1 can be easily extended towards other models that permit to reduce the worst-case latency (e.g., introducing the best-case execution time or non-preemptive regions may, in some cases, force the consumer task to execute always after the producer). As the method has exponential time complexity, we are currently working on finding an upper bound with linear time complexity that may be derived from the closed-form expression given in Lemma 1.

Acknowledgements

This research has been partially funded by the French FUI Waruna project.

References

[1] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study," *Computer Science and Information Systems, ISSN: 1820-0214*, vol. 10, no. 1, pp. 453–482, January 2013.

[2] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Transactions on computers*, vol. 39, no. 9, pp. 1175–1185, 1990.

[3] J. Schlatow and R. Ernst, "Response-time analysis for task chains in communicating threads," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Vienna, Austria, April 2016, pp. 1–10.

[4] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, "Scheduling dependent periodic tasks without synchronization mechanisms," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, Stockholm, Sweden, April 2010, pp. 301–310.

[5] R. Wyss, F. Boniol, J. Forget, and C. Pagetti, "Propriétés de latence, fraîcheur et réactivité dans un programme synchrone multi-périodique," in *Approches Formelles dans l'Assistance au Développement de Logiciels*, Nancy, France, Apr. 2013.

[6] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *Proceedings of the IEEE Real-Time System Symposium – Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, Barcelona, Spain*, 2008.

[7] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proceedings of the 44th annual Design Automation Conference*. San Diego, USA: ACM, 2007, pp. 278–283.

[8] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing end-to-end timing constraints by calibrating intermediate processes," in *Proceedings of the Real-Time Systems Symposium*, San Juan, Puerto Rico, USA, Dec 1994, pp. 192–203.

[9] ——, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Transactions on Software Engineering*, vol. 21, no. 7, pp. 579–592, Jul 1995.

[10] M. D. Natale, P. Giusto, S. Kanajan, C. Pinello, and P. Popp, "Architecture Exploration for Time-Critical and Cost-Sensitive Distributed Systems," in *SAE Technical Paper*. SAE International, April 2007.

[11] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh, "Schedulability and end-to-end latency in distributed ecu networks: Formal modeling and precise estimation," in *Proceedings of the 10th ACM International Conference on Embedded Software*, ser. EMSOFT '10. Scottsdale, USA: ACM, 2010, pp. 129–138.

[12] V. Anwikar and P. Bhaduri, "Timing analysis of real-time embedded systems using model checking," in *Proceedings of the 18th International Conference on Real-Time and Network Systems*, Toulouse, France, 2010, pp. 119–128.

[13] J. Khatib, A. Munier-Kordon, E. C. Klikpo, and K. Trabelsi-Colibet, "Computing latency of a real-time system modeled by synchronous dataflow graph," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. Brest, France: ACM, 2016, pp. 87–96.

[14] R. Wyss, F. Boniol, C. Pagetti, and J. Forget, "End-to-end latency computation in a multi-periodic design," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ser. SAC '13. Coimbra, Portugal: ACM, 2013, pp. 1682–1687.

[15] J. Forget, F. Boniol, D. Lesens, and C. Pagetti, "A real-time architecture design language for multi-rate embedded control systems," in *Proceedings of the ACM Symposium on Applied Computing*, ser. SAC'10. Sierre, Switzerland: ACM, 2010, pp. 527–534.

[16] J. Forget, F. Boniol, and C. Pagetti, "Verifying end-to-end real-time constraints on multi-periodic models," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2017, pp. 1–8.

[17] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proceedings of the 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Daegu, Korea, Aug 2016, pp. 159–169.

[18] ——, "Mechaniser-a timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies," in *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems WATERS'16*, Toulouse, France, 2016.

[19] ——, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, no. Supplement C, October 2017.

[20] ——, "Analyzing end-to-end delays in automotive systems at various levels of timing information," *ACM SIGBED Review: Special Issue on 4th International Workshop on Real-time Computing and Distributed Systems in Emergent Applications*, vol. 14, no. 4, pp. 1–6, November 2017.

[21] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, no. 2-3, pp. 117–134, Apr. 1994.

[22] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," in *Real-Time Systems Symposium., 17th IEEE*, Washington DC, USA, Dec 1996, pp. 22–29.

[23] D. Clark, "Hic: an operating system for hierarchies of servo loops," in *Proceedings, International Conference on Robotics and Automation*, vol. 2, Scottsdale, USA, May 1989, pp. 1004–1009.

[24] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication Centric Design in Complex Automotive Embedded Systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 76, Dubrovnik, Croatia, 2017, pp. 10:1–10:20.

[25] J. Y.-T. Leung and M. Merrill, "A note on preemptive scheduling of periodic, real-time tasks," *Information Processing Letters*, vol. 11, no. 3, pp. 115 – 118, 1980.

[26] Y. Xu, A. Cervin, and K. E. Årzén, "Harmonic scheduling and control co-design," in *Proceedings of the 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Daegu, Korea, Aug 2016, pp. 182–187.

[27] M. Mohaqeqi, M. Nasri, Y. Xu, A. Cervin, and K.-E. Årzén, "On the problem of finding optimal harmonic periods," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. Brest, France: ACM, 2016, pp. 171–180.

[28] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmark for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Lund, Sweden, July 2015.

[29] E. Bini and G. Buttazzo, "Biasing effects in schedulability measures," in *16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004. Proceedings*, 2014, pp. 196 – 203.

[30] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, ser. WATERS, 2014.