This is a pre print version of the following article:

Analytical Characterization of End-to-End Communication Delays With Logical Execution Time / Martinez, J.; Sanudo, I.; Bertogna, M.. - In: IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. - ISSN 0278-0070. - 37:11(2018), pp. 2244-2254. [10.1109/TCAD.2018.2857398]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

29/03/2024 12:26

(Article begins on next page)

Analytical Characterization of End-to-End Communication Delays with Logical Execution Time*

Authors omitted for blind review

ABSTRACT

Modern automotive embedded systems are composed of multiple real-time tasks communicating by means of shared variables. The effect of an initial event is typically propagated to an actuation signal through sequences of tasks writing/reading shared variables, creating an *effect chain*. The responsiveness, performance and stability of the control algorithms of an automotive application typically depend on the propagation delays of selected effect chains. Indeed, task jitter can have a negative impact on the system potentially leading to instability. The Logical Execution Time (*LET*) model has been recently adopted by the automotive industry as a way of reducing jitter and improving the determinism of the system.

In this paper, we provide a formal analysis of the LET model for real-time systems composed of periodic tasks with harmonic and non-harmonic periods, analytically characterizing the control performance of LET effect chains. We also show that by introducing tasks offsets, the real-time performance of non-harmonic tasks may improve, getting closer to the constant end-to-end latency experienced in the harmonic case. Further, we present a heuristic algorithm to obtain a set of offsets that might reduce end-to-end latencies, improving LET communication determinism. Finally, we apply this technique to an industrial case study consisting of an automotive engine control system.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference Format:

Authors omitted for blind review. 2018. Analytical Characterization of End-to-End Communication Delays with Logical Execution Time. In *Proceedings of Embedded systems week (EMSOFT),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

In the AUTOSAR[1] model, the typical way tasks communicate is through shared variables, i.e., labels, that are written/read by two or more runnables. Different communication patterns are used in the automotive industry to ensure a consistent communication between tasks, each having a different impact over the communication latencies experienced by tasks accessing the same shared variable [8][1].

Automotive applications are particularly concerned with optimizing end-to-end propagation latencies of input events that trigger a chain of computations, leading to a final actuation or control action. An *effect chain* (*EC*) is defined as a chain of reading/writing operations, typically triggered by a given event, where a task writes a label, which is then read by a second task; this latter task processes the read variable, and then writes a different label, which is then read by a third task. And so on, until the end of the chain. Usually, each chain is associated to given timing constraints that reflect the dynamics of the controlled system. The amount of time that elapses from the first input event until the end of the chain may significantly affect the control performance of the considered application [17][15].

Lately, there has been an increasing interest in the LET model in industrial domains, such as automotive [9] and avionics [25] [11], thanks to the improved determinism that can be achieved. In a real-time context, the LET semantics fixes the time it takes from reading task input to writing task output, regardless of the actual execution time of the task. Due to its semantics, the LET communication may lengthen the end-to-end latency of an effect chain in comparison to other communication patterns [1]. Moreover, if the effect chain is composed of tasks with harmonic periods, then the end-to-end latency is always constant. However, if one pair has non-harmonic periods, then the end-to-end latency may vary due to the misalignment of the task periods. We therefore seek a method that aims at reducing this misalignment, and so shortens, and might even stabilize, the end-to-end latency of an EC that obeys the LET semantics.

Offset assignment [24] is a well-known technique that has been adopted in the past to reduce the output jitter of a task, interact with slow devices, establish precedence constraints, obtain resource separation, increase feasibility bounds, and shorten worst-case response times (*WCRT*) [2]. Static and dynamic offset assignment has also been studied in the context of multiprocessor and distributed systems [21]. Recently, there has been a revival of interest in this technique to achieve efficient and effective non-preemptive scheduling by using a First-In-First-Out (*FIFO*) scheduling policy [19].

In this paper, we show that communication determinism may be improved by combining static offset assignment with the LET model. To that end, we present a novel heuristic algorithm to assign task offsets to reduce not only task WCRTs, but also end-to-end latency and jitter. We show that the proposed algorithm may achieve comparable performance of a brute force method that explores the

*Produces the permission block, and copyright information

[1] https://www.autosar.org/

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

EMSOFT, 2018, Torino, Italy

© 2018 Copyright held by the owner/author(s).

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

whole design space, but with a much more reasonable computational complexity.

The paper is organized as follows. The following section introduces the rationale behind the use of LET in the automotive domain. Section 3 presents the state-of-the-art with relation to the LET paradigm, the offset-based analysis for static priority task systems, and offset assignment methods. Section 4 presents our scheduling model, as well as the related response-time analysis. Section 5 formally presents the LET model and introduces the concept of publishing and reading points. Section 6 derives an exact end-to-end analysis of tasks obeying the LET semantics, presenting the advantages of an offset-aware LET model. A heuristic algorithm is then presented in section 7 to compute a set of offsets that improves real-time performance and control determinism. An experimental characterization of our heuristics is presented in section 8 using an automotive industrial case study consisting of an engine control system provided by Bosch [9]. Finally, section 9 presents our conclusions and directions for future works.

2 MOTIVATION

In an AUTOSAR application for the automotive domain, the smallest functional entity is called *runnable*. Runnables having the same functional period based on control dynamics are typically grouped into the same task. In the simplest case, one functionality is realized by means of a single runnable. Nevertheless, more complex functionalities are typically accomplished using several communicating runnables, possibly distributed over multiple tasks. Given an existing operational system, new functionalities are typically added by the addition or replacement of runnables, potentially modifying task computation times. These modifications may have a big impact on the end-to-end latency of a given effect chain.

Consider the example in Figure 1, where an effect chain composed of τ_1, τ_2 and τ_3 is shown. Task τ_1 has a runnable writing a label that is then read by τ_2; this latter task processes the read variable, and then writes a different label, which is then read by a runnable in τ_3. In the end, this runnable outputs an actuation signal that completes the effect chain. In this case, the amount of time that elapses from the first input event until the end of the chain, also known as the end-to-end latency, is 3. If the computation time of some runnables is modified, or more runnables are added as in Figure 2, the end-to-end latency may increase (19 for the case in the figure).

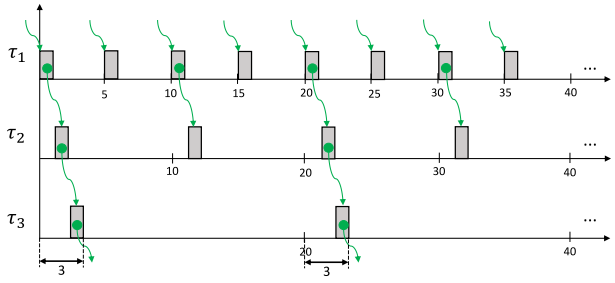


Figure 1: End-to-end effect chains composed of three tasks with parameters $T_1 = 5, T_2 = 10, T_3 = 20$ and $C_1 = C_2 = C_3 = 1$.

Control tasks are typically executed periodically, i.e., at a given sampling period. The resulting control performance is highly dependent on task jitter, task response times, scheduling policy and end-to-end latency of effect chains. Even a small change in one of these parameters might be detrimental to control performance, potentially requiring a system redesign, with related additional cost and time.

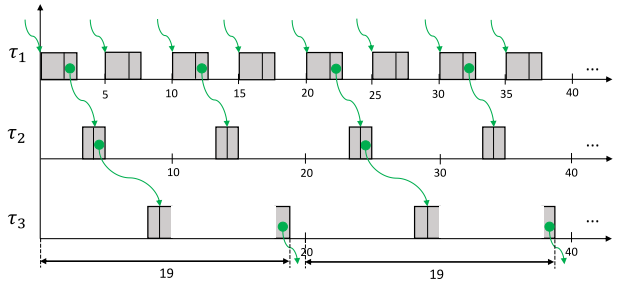


Figure 2: End-to-end effect chains composed of three tasks with parameters $T_1 = 5, C_1 = 3, T_2 = 10, C_2 = 2, T_3 = 20$ and $C_3 = 3$.

Even with constant execution times, different instances of the same task might have different response times, leading to variable end-to-end latencies of an effect chain. An example is shown in Figure 3. The LET concept has been introduced in the automotive industry to explicitly address this issue. The LET semantics decouples control algorithms from task jitter, task response times, scheduling policy and hardware dependence, enabling more robust algorithms and more deterministic and predictable systems, as explained in section 5.

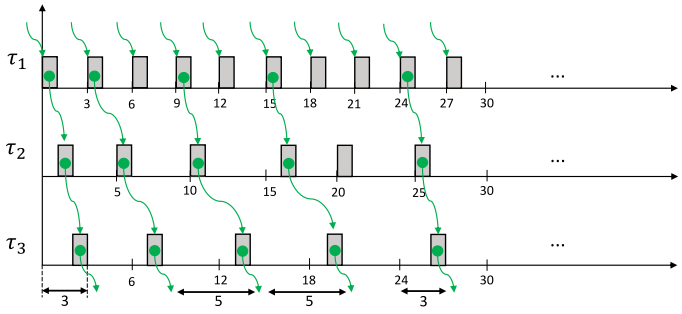


Figure 3: End-to-end effect chains composed of three tasks with parameters $T_1 = 3, T_2 = 5, T_3 = 6$ and $C_1 = C_2 = C_3 = 1$.

3 RELATED WORK

The Logical Execution Time (LET) paradigm has been proposed within the time-triggered programming language Giotto [10]. This communication pattern allows determining the time it takes from reading program input to writing program output, regardless of the actual execution time of a real-time program. As stated in [12], LET evolved from a highly controversial idea to a well-understood principle of real-time programming, motivated by the observation

that the relevant behavior of real-time programs is determined by when inputs are read and outputs are written. This concept has been adopted by the automotive and avionics industry as a way of introducing determinism in their systems.

In [8], an overview of the different communication patterns adopted in the automotive domain is provided, highlighting the importance of end-to-end latency of effect chains in an engine management system. A method to transform LET into a corresponding direct communication is also presented, allowing the use of classic tools (e.g., SymTA/S[2]) to determine end-to-end latencies and communication overhead. In [3], an end-to-end timing latency analysis for effect chains with specified age-constraints is presented. The analysis is based on deriving all possible data propagation paths which are used to compute the minimum and maximum end-to-end latency of effect chains. In [4], the analysis is extended to include the Logical Execution Time paradigm, providing an algorithm to derive the maximum data age of cause-effect chains. However, none of these works takes offset assignment into consideration.

As previously mentioned, offset assignment is a well-known method to reduce the output jitter of tasks, improving system schedulability and shortening the WCRT of tasks. A proper selection of task offsets may increase the predictability of the system by better distributing the workload over time. In [24], Tindell introduced the idea of using task offsets to model periodic transactions of different tasks. An exact response time analysis (RTA) was proposed for tasks with static offsets, showing that offsets can be used to reduce the pessimism of the classic response time analysis. Unfortunately, the presented RTA is computationally intractable but for small tasks sets. Therefore, an approximate RTA was also proposed. Later on, Palencia and Harbour [20] extended the approximate RTA of Tindell by analyzing tasks with static and dynamic offsets for distributed systems. While the static analysis assumes that offsets are fixed from the transaction release, dynamic offset analysis considers that offsets may change from one activation to another. In [23], a method is described to perform exact RTA for fixed priority tasks with offsets and release jitter based on the work in [20]. Recently, a RTA aware of end-to-end timing requirements has been published by Palencia et al. [22]. In this work, a method is presented to perform an offset-based RTA for time-partitioned distributed systems. Authors also considered effect chains with precedence constraints.

In [6], Goossens distinguished between three types of periodic task sets: (i) synchronous, where the offsets are fixed and all equal to 0 ($O_1 = O_2 = \ldots = O_n = 0$); (ii) asynchronous, where offsets are determined by the constrains of the system; and (iii) offset-free, where offsets are chosen by the scheduling algorithm. A method to assign offsets is presented, proposing different heuristics to determine a static offset for each task.

The offset assignment problem has also been studied for the automotive domain. In [7], Grenier et al. proposed the use of offsets to improve the task schedulability of body and chassis networks considering CAN-bus related delays. This technique is used to minimize the WCRT by distributing the workload over time. An offset assignment algorithm tailored for automotive CAN networks is presented to improve task WCRT. Based on this algorithm, Monot et al. proposed in [18] runnable-to-task allocation heuristics for multicore platforms, balancing the CPU load over the system through offset assignment. Recently, Nasri et. al [19] presented an offset assignment technique for FIFO scheduling in order to obtain schedulability performance comparable to non-preemptive fixed priority scheduling, while incurring a smaller overhead.

To the best of our knowledge, the present work is the first study that formally defines an offset-aware schedulability analysis for the LET model. The impact of an offset-aware LET model on the end-to-end latency of effect chains is thoroughly analyzed, proposing a heuristic algorithm to obtain a convenient offset assignment.

[2]https://auto.luxoft.com/uth/timing-analysis-tools/

4 SYSTEM MODEL AND NOTATION

This section describes the terminology and notation used throughout the paper.

We assume a system composed of m identical cores, with periodic tasks and runnables statically partitioned to the cores, using any given scheduler with no task migration support. Each task τ_i is characterized by a tuple (T_i, C_i, O_i), where T_i is the period, C_i is the worst-case execution time (WCET) and O_i is the initial offset. Deadlines are assumed to be equal to periods. Each task τ_i releases an infinite sequence of jobs, with the first job released at time O_i, and subsequent jobs periodically released at time $r_{i,k} = O_i + kT_i$. Without loss of generality, we assume $O_i < T_i$ for all tasks τ_i.

The *hyperperiod* of the task system is the least common multiple of the task periods. In case of a fixed priority scheduler, the worst-case response time R_i of a task τ_i with offset can be computed taking the largest response time of all the jobs released by τ_i in a hyperperiod, as described in [23]. In this paper, we are interested in task sets that are schedulable independently of the offset, i.e. for any task τ_i: $R_i \leq T_i, \forall O_i$. For the fixed priority case, this means considering tasks that are schedulable in the synchronous periodic case, i.e., when all offsets are null, which represents a critical instant scenario [16].

A task can be either a writer or a reader of a label. We assume there is only one writer per label, while there may be multiple readers reading that label. All parameters are integer multiples of the system clock.

5 LOGICAL EXECUTION TIME

In the context of hard real-time systems, the LET semantics enforces task communications at deterministic times, corresponding to task activation times. LET fixes the time it takes from reading task input to writing task output, regardless of the actual execution time of the task. Inputs and outputs are logically updated at the beginning and at the end of their LET, respectively, see Figure 4. In this paper we assume that the LET equals the task period. It is worth mentioning that the LET paradigm assumes these updates incur zero computation time. In [1] an implementation is presented that emulates this ideal behavior by making use of buffers in order to guarantee the determinism of the communication.

We hereafter consider the communication between the writer and one of the readers. Assume the writer and the reader have period $T_W = 2$ and $T_R = 5$, respectively, as in Figure 5. While τ_W may repeatedly write the considered labels, these updates are not visible to the concurrently executing reader, until a *publishing point*

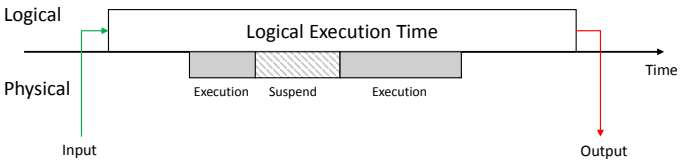


Figure 4: Logical Execution Time model.

$P^n_{W,R}$, where the values are updated for the next reader instance. This point corresponds to the first upcoming writer release that directly precedes a reader release, i.e., where no other write release appears before the arrival of the following reader instance. We call *publishing instance* the writing instance that updates the shared values for the next reading instance, i.e., the writer's job that directly precedes a publishing point. Note that not all writing instances are publishing instances. See Figure 5, where publishing instances are marked in bold red.

It is also convenient to define *reading points* $Q^n_{R,W}$, which correspond to the arrival of the reading instance that will first use the new data published in the preceding publishing point $P^n_{R,W}$. Figure 6 shows publishing and reading points for a case where $T_W = 5$ and $T_R = 2$.

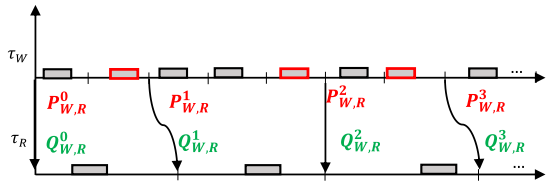


Figure 5: Publishing and reading points when the reader has larger period than the writer.

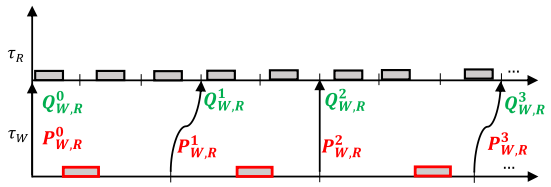


Figure 6: Publishing and reading points when the reader has smaller period than the writer.

The publishing and reading points of two communicating tasks can be computed as a function of their periods, as shown in the next theorem.

Theorem 5.1. *Given two communicating tasks τ_W and τ_R, the publishing and the reading points can be computed as*

$$P^n_{W,R} = \left\lfloor \frac{nT_{\max}}{T_W} \right\rfloor T_W \tag{1}$$

$$Q^n_{W,R} = \left\lceil \frac{nT_{\max}}{T_R} \right\rceil T_R \tag{2}$$

where $T_{\max} = \max(T_W, T_R)$

Proof. If the writer τ_W has a smaller or equal period than the reader τ_R, i.e., $T_W \leq T_R$ as in Figure 5, there is one publishing and one reading point for each *reading* instance. Reading points trivially correspond to each reading task release, i.e.,

$$Q^n_{W,R} = nT_R,$$

while publishing points correspond to the last writer release before such a reading instance, i.e.,

$$P^n_{W,R} = \left\lfloor \frac{nT_R}{T_W} \right\rfloor T_W.$$

Otherwise, when the writer τ_W has a larger period than the reader τ_R, i.e., $T_W \geq T_R$ as in Figure 6, there is one publishing and one reading point for each *writing* instance. Publishing points trivially correspond to each writing task release, i.e.,

$$P^n_{W,R} = nT_W,$$

while reading points correspond to the last reader release before such a writing instance, i.e.,

$$Q^n_{W,R} = \left\lceil \frac{nT_W}{T_R} \right\rceil T_R.$$

It is easy to see that, in both cases $T_W \leq T_R$ and $T_W \geq T_R$, the formulas for $P^n_{W,R}$ and $Q^n_{W,R}$ are generalized by Equations (1) and (2). Note that, when $T_W = T_R$, $P^n_{W,R} = Q^n_{W,R} = nT_W$. □

Two communicating tasks τ_W and τ_R have harmonic periods if the period of one of them is an integer multiple of the other. When a harmonic synchronous communication (HSC) is established, the following relations hold: $LCM(T_W, T_R) = T_{\max}$ and $P^n_{W,R} = Q^n_{W,R} = nT_{\max}$, i.e., publishing and reading points are integer multiples of the largest period of the communicating tasks. On the other hand, when two communicating tasks do not have harmonic periods, a non-harmonic synchronous communication (NHSC) is established. The general formulas of Theorem 5.1 apply.

6 END-TO-END LATENCY ANALYSIS

An effect chain is a producer/consumer relationship between runnables working on labels. As mentioned in the introduction, effects chains are assumed to be triggered by an external event or a task release. The first task in the chain produces an output (i.e., writes to a label) for another task following in the event chain. The second task reads the label to write an output to a different label, which may be then read by a third task, and so on. When the last task produces its final output, the event chain is over. See Figure 1, 2 and 3.

In [5], four different end-to-end timing semantics are described to characterize the timing delays of effect chains given by multi-rate tasks communicating by means of shared variables. Depending on the application requirements, different end-to-end delay metrics can be of interest. Control systems driving external actuators are interested in the *age* of an input data, i.e., for how long a given sensor data will be used to take actuation decisions. For example, how long a radar or camera frame will be used as a valid reference by a localization or object detection system to perceive the environment: the older the frame, the less precise the system. Similar considerations are valid for an engine control or a fuel injection system, where correct actuation decisions depend on the *freshness* of sensed data.

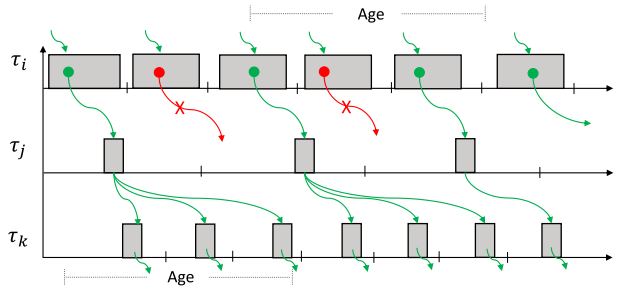


Figure 7: Age latency of an effect chain composed of three tasks.

Another metric of interest is the *reaction* latency to a change of the input, i.e., how long does it take for the system to react to a new sensed data. Multiple body and chassis automotive applications are concerned with this metric. For example, for a door locking system, it is important to know the time it takes to effectively lock the doors after receiving the corresponding signal. Due to space constraints, in this work we only cover age latency. However, similar results apply also for reaction latency.

To more formally characterize age latency, consider Figure 7, showing an event chain triggered by a periodic sensor. The upper task reads the sensor data, elaborates it, and shares the result with the next task. And so on, until the end of the event chain. Green arrows denote when an input is propagated to the next task. In this case, we call it a *valid* input. Red arrows correspond to elaborations that are not propagated, also called *invalid* inputs, because they are overwritten before being read by the next task in the chain. The *age latency* is defined as the delay between a valid sensor input until the last output related to this input in the event chain. It measures for how long an input continues influencing the final output of the event chain. In [5], age latency is also referred to as last-to-last (L2L). However, no method is presented to formally compute these metrics.

As discussed in the previous section, the LET model requires that inputs and outputs be logically updated at reading and publishing points, respectively. To see its effect on end-to-end latency, let's apply its semantics to the examples shown in Figure 1 and 2. The results are shown in Figure 8, where it is easy to see that the age latency is the same in both cases. Clearly, this communication pattern allows not only deterministically setting publishing and reading points, but also setting the age latency of an effect chain to a fixed value, regardless of the actual execution time and core allocation of the involved communicating tasks. In this way, it is possible to achieve a higher level of predictability and a stronger consistency between the timing constraints (logical model) and the task execution (physical model), thus facilitating the design, implementation, test and certification process [13].

However, in the NHSC case, the above property does not hold. Consider the example shown in Figure 9a, end-to-end latencies are either 18 or 21, with a worst-case age latency of 21. However, assigning an offset of 1 to τ_3, as depicted in Figure 9b, reduces the worst-case age latency to 19, with zero jitter. This shows that by

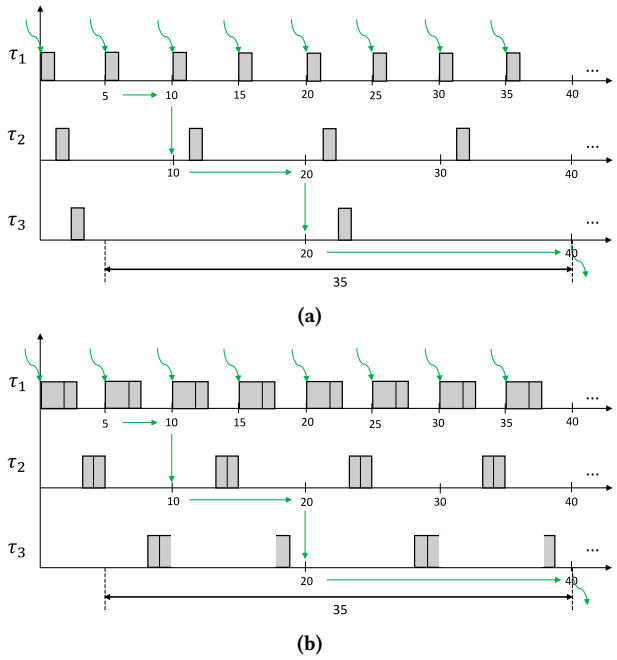


Figure 8: End-to-end effect chain with LET composed of three tasks with parameters: $T_1 = 5, T_2 = 10, T_3 = 20$ **with (a)** $C_1 = C_2 = C_3 = 1$ **and (b)** $C_1 = 3, C_2 = 2, C_3 = 3$**.**

properly assigning offsets it is possible to improve control performance of NHSC, reducing the predictability gap in comparison with HSC by decreasing worst-case age latency and reducing jitter.

In order to understand how to properly assign offsets, we first generalize Theorem 6.1 to consider offsets.

Theorem 6.1. *Given two communicating tasks τ_W and τ_R, with offsets O_W and O_R, respectively, the publishing and the reading points can be computed as*

$$P^n_{W,R} = O_W + \left\lfloor \frac{nT_{\max} + O_{max} - O_W}{T_W} \right\rfloor T_W \tag{3}$$

$$Q^n_{W,R} = O_R + \left\lceil \frac{nT_{\max} + O_{max} - O_R}{T_R} \right\rceil T_R \tag{4}$$

where $T_{\max} = \max(T_W, T_R)$, and $O_{\max}$ is the offset of the task with the largest period in the pair.

Proof. The proof is very similar to that of Theorem 5.1. If the writer τ_W has a smaller or equal period than the reader τ_R, i.e., $T_W \leq T_R$ as in Figure 10, there is one publishing and one reading point for each *reading* instance. Reading points again correspond to each reading task release, this time including offset:

$$Q^n_{W,R} = O_R + nT_R,$$

while publishing points correspond to the last writer release before such a reading instance, i.e.,

$$P^n_{W,R} = O_W + \left\lfloor \frac{nT_R + O_R - O_W}{T_W} \right\rfloor T_W.$$

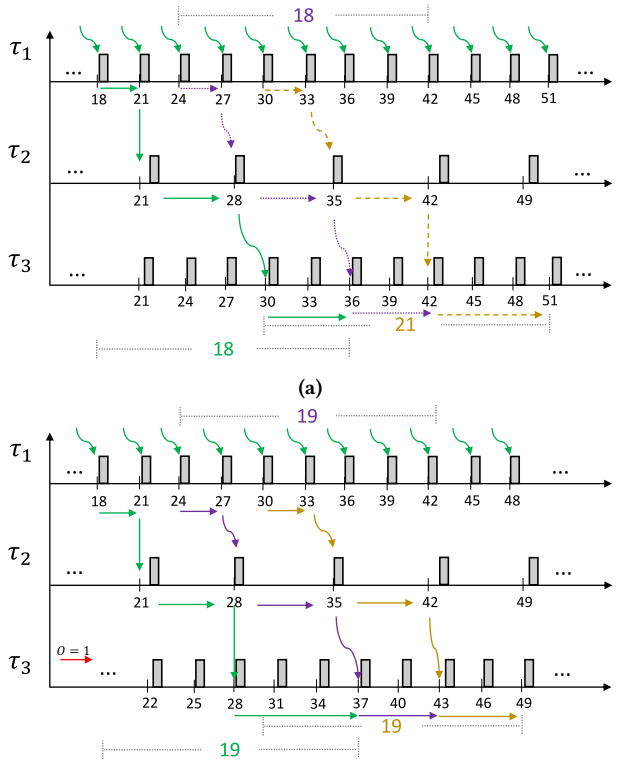


Figure 9: End-to-end effect chains with LET composed of three tasks with parameters (a) $T_1 = 3, O_1 = 0, T_2 = 7, O_2 = 0, T_3 = 3, O_3 = 0$ **with** $C_1 = C_2 = C_3 = 1$ **and (b)** $T_1 = 3, O_1 = 0, T_2 = 7, O_2 = 0, T_3 = 3, O_3 = 1$ **with** $C_1 = C_2 = C_3 = 1$

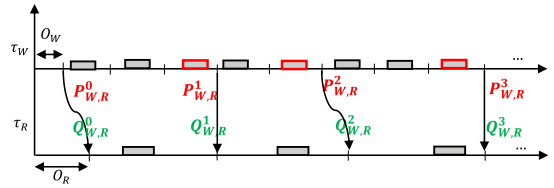


Figure 10: Publishing and reading points with offsets with $T_W = 2, O_W = 1, T_R = 5, O_R = 2$.

Otherwise, when the writer τ_W has a larger period than the reader τ_R, i.e., $T_W \geq T_R$ as in Figure 11, there is one publishing and one reading point for each *writing* instance. Publishing points correspond to each writing task release, including offset:

$$P^n_{W,R} = O_W + nT_W,$$

while reading points correspond to the last reader release before such a writing instance, i.e.,

$$Q^n_{W,R} = O_R + \left\lceil \frac{nT_W + O_W - O_R}{T_R} \right\rceil T_R.$$

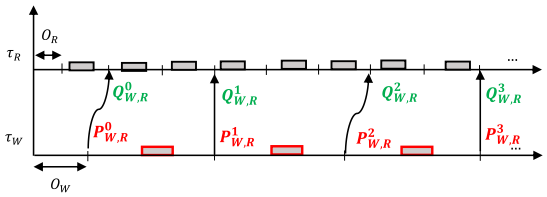


Figure 11: Publishing and reading points with offsets with $T_W = 5, O_W = 2, T_R = 2, O_R = 1$.

In both cases, the formula for $P^n_{W,R}$ and $Q^n_{W,R}$ are generalized by Equations (3) and (4). □

Clearly, the above theorem generalizes Theorem 5.1. When $T_W = T_R$, it can again be verified that each writing (resp. reading) task release correspond to a publishing (resp. reading) point.

Let us define the hyperperiod H_{EC} of an EC as the least common multiple of the periods of the tasks composing the chain, i.e., $H_{EC} = LCM_{i=1}^{\eta}(T_i)$, where η is the length of the EC, i.e., the number of tasks that compose the EC. Given all the publishing and reading points of the tasks composing an EC in its hyperperiod H_{EC}, we would like to compute the age latency of this chain. There is a fixed number of possible communication paths in H_{EC}. To characterize them, we define the notion of *basic path*, as an interval starting from the end of the period of the first task in the *EC*, and finishing with the release of the last task in the *EC*. For example, in the EC of Figure 12 there are three basic paths in the highlighted hyperperiod $H_{EC} = 21$: $[21, 30]$, $[27, 36]$ and $[33, 42]$. Note that if all tasks in the EC have harmonic periods, then there is only one basic path in the hyperperiod. In this case, the length of the basic path equals the sum of the periods of all tasks in the EC excluding the first task in the chain. In the examples of Figure 8a and 8b, there is only one basic path $[10, 20]$.

Let us define $\dot{P}^n_{W,R}$ (resp. $\dot{Q}^n_{W,R}$) as the publishing (resp. reading) point between two tasks τ_W and τ_R in the n-th basic path of an EC. Then, the n-th basic path in the EC starts at $\dot{P}^n_{1,2}$ and ends at $\dot{Q}^n_{\eta-1,\eta}$. See Figure 12. Note that $\dot{P}^n_{W,R}$ and $\dot{Q}^n_{W,R}$ are not necessarily equal to $P^n_{W,R}$ and $Q^n_{W,R}$.

Algorithm 1 shows how to compute the boundaries $\dot{P}^n_{1,2}$ and $\dot{Q}^n_{\eta-1,\eta}$ of the n-th basic path of an EC. Once these points are known, the length θ^n_{EC} of the n-th basic path of the EC can be simply computed as $\theta^n_{EC} = \dot{Q}^n_{\eta-1,\eta} - \dot{P}^n_{1,2}$.

In the following, we assume the η tasks of an EC be ordered according to their appearance in the considered effect chain, i.e., τ_1 is the first (writing) task in EC, while τ_η is the last (reading) task in EC. If we assume the EC is triggered by the release of the first task in the chain, the age latency α^n associated to the n-th basic path can then be computed by adding to the basic path length (i) the period T_1 of the first task in the EC, and (ii) the distance to the end of the next $(n+1)$-th basic path, where the output of the EC will eventually reflect a new input signal. That is,

$$\alpha^n = T_1 + \theta^n_{EC} + \dot{Q}^{n+1}_{\eta-1,\eta} - \dot{Q}^n_{\eta-1,\eta}. \quad (5)$$

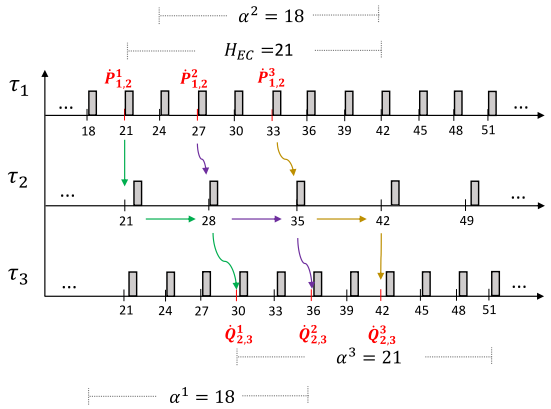


Figure 12: End-to-end effect chain characterization with LET composed of three tasks with parameters $T_1 = 3, O_1 = 0, T_2 = 7, O_2 = 0, T_3 = 3, O_3 = 0$.

The worst-case age latency $\alpha(EC)$ of the EC is then given by the maximum α^n over all basic paths in a hyperperiod of the EC.

$$\alpha(EC) = \max_{\forall n \in H_{EC}} \alpha^n. \tag{6}$$

Algorithm 1 Calculating the start and end of a basic path

1: *Input*: Task set in order of communication.
2: Group the tasks pairwise, i.e., $\{\tau_1, \tau_2\}$, $\{\tau_2, \tau_3\}$, ..., $\{\tau_{\eta-1}, \tau_\eta\}$.
3: Compute $\rho = \left\lceil 2 \cdot \sum_{i=1}^{\eta} T_i / H_{EC} \right\rceil + 1$.
4: Compute all the publishing and reading points of each pair for ρ hyperpriods H_{EC}.
5: **for** each $Q^n_{\eta-1,\eta}$ in the ρth-H_{EC} **do**
6: **for** i=η...3 **do**
7: Obtain the $P^n_{i-1,i}$ corresponding to $Q^n_{i-1,i}$.
8: Obtain the $Q^m_{i-2,i-1}$ preceding $P^n_{i-1,i}$.
9: n=m ; i=i-1
10: Given $Q^n_{1,2}$ get $P^n_{1,2}$.
11: Erase paths starting with the same publishing point $P^n_{1,2}$ of a previous path.

7 HEURISTICS

In the previous sections, we showed how an offset-aware LET analysis may be used to improve real-time performance. Nonetheless, it is worth mentioning that while offset assignment can shorten the age latency of a particular EC, it might also lengthen the end-to-end latency of another chain. On the other hand, effect chains, very much like tasks, are also prioritized, i.e. an EC might be of paramount importance to the stability and control of the system while another might not, and thus optimizing the end-to-end latency of this EC to the detriment of that of another might be necessary. For this reason, given a schedulable task set, we are interested in seeking an offset assignment method that shortens the age latency of an EC, possibly making it constant throughout the whole execution of the tasks involved. This could be particularly useful for automotive applications where there are no design constraints on offsets.

Without loss of generality, offsets can be normalized assuming $O_1 = 0$ and $O_i \in [0, T_i\rangle, \forall i \in [2, \eta]$. It is worth pointing out that the heuristics presented by Goossens in [6] cannot be applied, as it has a different target, i.e., making a task set schedulable, or reducing the worst-case response time of an already schedulable task set, on a uniprocessor. A brute force approach is not desirable for longer chains or when the periods of the tasks involved are large, since the number of combinations can get up to $\prod_{i=2}^{\eta} T_i = O((max_{j=2}^{\eta} T_j)^{\eta-1})$ for chains composed of different tasks. We therefore derive a heuristics for a convenient offset assignment that can be conveniently used to improve control performance within a reasonable computational complexity.

Equation 5 can be rewritten as

$$\alpha^n = T_1 + \dot{Q}^n_{\eta-1,\eta} - \dot{P}^n_{1,2} + \dot{Q}^{n+1}_{\eta-1,\eta} - \dot{Q}^n_{\eta-1,\eta} = T_1 + \dot{Q}^{n+1}_{\eta-1,\eta} - \dot{P}^n_{1,2}$$

From Theorem 6.1, it follows that

$$\alpha^n = T_1 + \left\lceil \frac{(n'+1)max(T_{\eta-1}, T_\eta) + O^{\eta-1,\eta}_{max} - O_\eta}{T_\eta} \right\rceil T_\eta + O_\eta - \left\lfloor \frac{n''max(T_1, T_2) + O^{1,2}_{max} - O_1}{T_1} \right\rfloor T_1 - O_1, \tag{7}$$

where $O^{i,j}_{max}$ is the offset of the task with the largest period among τ_i and τ_j, while n' and n'' are numbers defined by the alignment, periods and offsets of the tasks composing the n-th basic path of the EC. Let us define two integer values $k' = \left\lceil \frac{(n'+1)max(T_{\eta-1}, T_\eta) + O^{n-1,n}_{max} - O_\eta}{T_\eta} \right\rceil$ and $k'' = 1 - \left\lfloor \frac{n''max(T_1, T_2) + O^{1,2}_{max} - O_1}{T_1} \right\rfloor$. Then,

$$\alpha^n = k'T_\eta + k''T_1 + O_\eta - O_1 \tag{8}$$

Recalling that $O_1 = 0$,

$$\alpha^n = k'T_\eta + k''T_1 + O_\eta \tag{9}$$

The last equation shows that the age latency of an EC can be computed as the sum of a multiple of the period of the first and of the last task in the chain, plus the offset of the last task. This does not mean that the tasks in the middle of the chain have no influence on the age latency. Their contribution is hidden within k' and k'', which may increase or decrease the age latency by integer multiples of the period of the first and last task in the EC.

The fact that the offset of the final task in the chain, O_η, appears in the previous equation forms the basis for Algorithm 2. This algorithm proposes a heuristic approach to assign offsets that considers only the last d tasks in the EC, starting from the last task τ_η. The remaining $\eta - d$ tasks are assumed to have a null offset. In this way, the total number of combinations is reduced to $\prod_{i=\eta-d+1}^{\eta} T_i = O((max_{j=\eta-d+1}^{\eta} T_j)^d)$. Note that $d < \eta$ and $O_1 = 0$. Furthermore, $d = \eta - 1$ is equivalent to the brute force approach.

The complexity can be further reduced by considering only non-equivalent offset assignments. Two asynchronous situations are defined to be equivalent, if they have the same periodic behavior.

For two tasks τ_1 and τ_2, two choices O_2 and O_2' are equivalent if they produce the same relative phasing, i.e.,

$$\exists k \in \mathbb{N} : O_2 \mod T_1 = (O_2' + kT_2) \mod T_1.$$

As an example, consider τ_1 and τ_2 with $T_1 = 8$, $T_2 = 12$, $O_1 = 0$ and $O_2 < T_2$. The offset assignment $O_2 = 0$ is equivalent to $O_2' = 4$ and to $O_2'' = 8$, since they all lead to the same job interleaving throughout the hyperperiod $LCM(T_1, T_2) = 24$. Similarly, $O_2 = 1$ is equivalent to $O_2' = 5$ and to $O_2'' = 9$. In general, two offset assignments $O2$ and $O2'$ are equivalent if $O_2 = O_2' \mod GCD(T1, T2)$, as shown in [6]. Therefore, it makes sense to consider only the offsets in $[0, GCD(T_1, T_2)\rangle$.

For later tasks in the effect chain, similar considerations apply by considering their alignment with respect to the hyperperiod of earlier tasks. E.g., for task τ_3, it is sufficient to consider its non-equivalent alignments with respect to the hyperperiod of τ_1 and τ_2, i.e., $O_3 \in [0, GCD\{T_3, LCM(T_1, T_2)\}\rangle$. In general, assuming the offsets $O_1, \ldots, O_{i-1}$ have been set, for τ_i it is sufficient to consider

$$O_i \in [0, GCD\{T_i, LCM_{j=1}^{i-1} T_j\}\rangle, \forall i \in [2, \eta]$$

Thus, the number of possible combinations of the brute force approach is reduced to

$$\prod_{i=2}^{\eta} GCD\left\{T_i, LCM_{j=1}^{i-1} T_j\right\}.$$

Since $x \cdot y = GCD(x, y) \cdot LCM(x, y)$, this simplifies to

$$\prod_{i=2}^{\eta} \frac{T_i \cdot LCM_{j=1}^{i-1} T_j}{LCM(T_i, LCM_{j=1}^{i-1} T_j)} = \prod_{i=2}^{\eta} \frac{T_i \cdot LCM_{j=1}^{i-1} T_j}{LCM_{j=1}^{i} T_j} = \frac{\prod_{i=1}^{\eta} T_i}{LCM_{i=1}^{\eta} T_i}.$$

The complexity of the brute force approach is then $\prod_{i=1}^{\eta} T_i / H_{EC}$. This entails a significant reduction in the complexity, especially in case of mutually prime periods. Note that in case all periods are mutually prime, there is only one configuration to check.

Similarly, the number of offset assignments leading to non-equivalent asynchronous situations given by the d-offset assignment algorithm can be derived as

$$\prod_{i=\eta-d+1}^{\eta} GCD\left\{T_i, LCM_{j=1}^{i-1} T_j\right\} = \prod_{i=\eta-d+1}^{\eta} \frac{T_i \cdot LCM_{j=1}^{i-1} T_j}{LCM(T_i, LCM_{j=1}^{i-1} T_j)}$$

$$= \prod_{i=\eta-d+1}^{\eta} \frac{T_i \cdot LCM_{j=1}^{i-1} T_j}{LCM_{j=1}^{i} T_j} = \frac{LCM_{i=1}^{\eta-d} T_i \cdot \prod_{i=\eta-d+1}^{\eta} T_i}{LCM_{i=1}^{\eta} T_i}.$$

Let $H_d = H_{EC} / LCM_{i=1}^{\eta-d} T_i$. The complexity of the d-offset assignment algorithm is then $(\prod_{i=\eta-d+1}^{\eta} T_i)/H_d$.

Algorithm 2 d-Offset assignment

1: *Input*: Task set $\{\tau_i\}$, depth d
2: Assign $O_i = 0, \forall i \in [1, \eta - d]$
3: Consider all combinations of offset assignments leading to non-equivalent asynchronous situations $\forall \tau_i, i \in [\eta - d + 1, \eta]$
4: **for** `each combination` **do**
5: Compute the worst-case age latency of this combination using Equation (6)
6: Return the maximum age latency among all combinations

8 EXPERIMENTS

Having established a thorough analytical characterization of the end-to-end latencies of effect chains under the Logical Execution Time communication model, we hereafter provide an experimental characterization of the effectiveness of LET in improving the control performance by reducing the variability of the end-to-end latency. Moreover, we show how the proposed offset assignment technique can be adopted to further reduce such a variability in case an even tighter control performance is needed.

To this end, we performed two sets of experiments. The first set considers an industrial case study from the automotive domain, providing a characterization of the analytical performance of LET in a representative setting. The second set of experiments is based on randomly generated effect chains composed of tasks with a different period distribution, to characterize the effectiveness of the offset assignment methods in further reducing jitter.

8.1 Industrial case study

To provide a representative characterization of the end-to-end latencies introduced by LET, we considered an automotive application representing an engine control systems, as detailed by Kramer et al. in [14]. The application is composed of multiple tasks partitioned onto four cores. The periods of the tasks are {1, 2, 5, 10, 20, 50, 100, 200, 1000}ms. Tasks are composed of 1250 runnables that access about 1500 different labels. We considered the effect chains created by tasks reading/writing a common shared variable. Based on this setting, there are over 500 ECs with length $3 \leq \eta \leq 8$.

Figure 13 shows the average value of the worst-case age latency $\alpha(EC)$ obtained with LET among the considered effect chains for each EC length. As can be expected, the age latency increases proportionally with the length of the chain. An analysis on the individual EC shows that the worst-case age latency is never smaller than the sum of the periods of the tasks composing the considered EC.

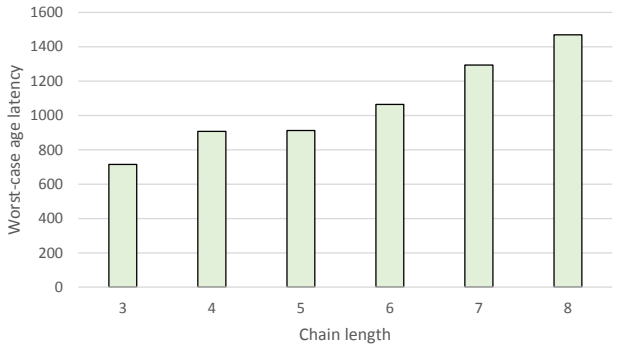


Figure 13: Average value of the worst-case age latency for the considered effect chains.

More interestingly, the LET model allows significantly reducing the jitter of the end-to-end latency of an effect chain. We define the jitter of an EC as

$$J(EC) = \max_{\forall n \in H_{EC}} \alpha^n - \min_{\forall n \in H_{EC}} \alpha^n.$$

Figure 14 shows the normalized jitter ($J(EC)/\alpha(EC)$), i.e., the ratio of the jitter over the age latency. Both average and worst-case values over all effect chains are shown for each considered length. The average jitter is always below 1%, confirming that LET is very effective in reducing end-to-end latency variability, with longer chains exposing a slightly smaller normalized jitter. However, for all considered EC lengths, there are different cases where the jitter is above 10% of the overall age latency.

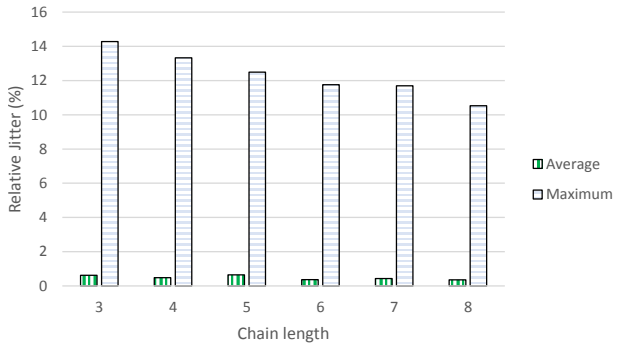


Figure 14: Average and maximum values of the normalized jitter for the considered effect chains.

In order to further improve the end-to-end control performances, we applied the offset assignment method of Algorithm 2. Even using a small depth $d = 1$ (resp. $d = 2$) allowed improving the worst-case age latency for 206 (resp. 377) out of the 577 considered effect chains. The improvement obtained for these ECs is shown in Figure 15 both for $d = 1$ and $d = 2$. In general, a small depth allows significantly improving the age latency of shorter chains (10% on average, 30% in the best case). A larger depth value allows improving the latency of longer chains, by paying a higher computational cost.

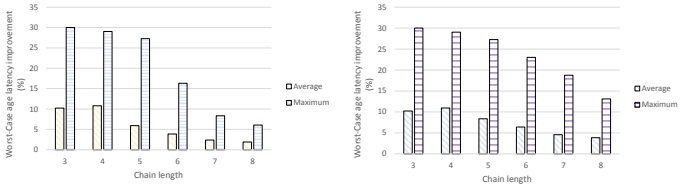


Figure 15: Average and maximum age latency improvement provided by the offset assignment heuristics with depth $d = 1$ (left) and $d = 2$ (right).

Another interesting effect of the offset assignment technique is to decrease the jitter. Note that effect chains composed of harmonic tasks have all a null jitter. In the considered automotive use case, the great majority of effects chains are harmonic, due to the selection of task periods. Therefore, the average and maximum jitter shown in Figure 14 is due to a few non harmonic effect chains, 32 of which had a non null jitter. With our offset assignment method, the jitter is reduced to zero for 9 of them with $d = 1$. Figure 16 shows the average and best-case improvement in the jitter normalized with respect to the age latency, i.e., $\Delta J(EC)/\alpha(EC)$, for the case with $d = 2$.

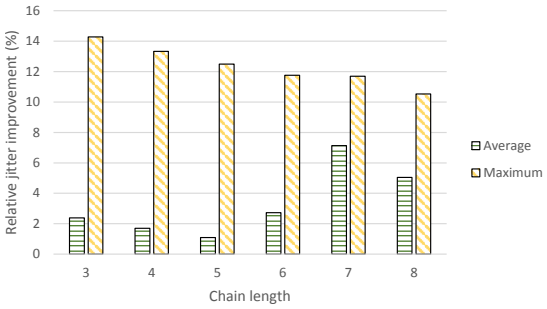


Figure 16: Average and maximum normalized jitter improvement provided by the offset assignment heuristics with depth $d = 2$

8.2 Randomly generated workloads

A second set of experiments is provided to characterize the efficiency of the proposed heuristics with respect to a brute force approach. Unfortunately, the industrial use case adopted in the previous section is not amenable to a brute force approach because of the large range of task periods, which makes it too computationally expensive. Therefore, we synthetically generated 500 effect chains composed of randomly generated tasks with periods uniformly distributed in $[1, 10]$. We considered effect chains with $\eta \in [3, 6]$. Note that there is no need to generate utilizations and execution times, since tasks are assumed to always complete before their (implicit) deadlines, as stated in section 4.

To understand the performance of the proposed heuristics in exploring the design space to select an optimal offset assignment, we provide a characterization based on the depth d value that allows achieving an optimal end-to-end latency. In this experiment, we first computed the optimal offsets using a brute force approach. Then, we ran Algorithm 2 with increasing depth values, starting with $d = 1$, to compare the resulting worst-case age latency with that of the brute force algorithm. When they matched, the algorithm was stopped recording the d value. Figure 17 shows the normalized depth r, defined as the ratio between the resulting d and the length of the EC, i.e., $r = d/\eta$. Interestingly, an optimal assignment is obtained even with a very small depth. In more than 60 % of the cases, r is lower than or equal to 1/3, indicating that the proposed heuristics can be conveniently adopted to reduce age latencies even using a small depth d.

9 CONCLUSION

In this paper, we provided an analytical characterization of the end-to-end latency of effect chains composed of periodic tasks communicating using the Logical Execution Time model. A closed formula expression was provided to compute reading and publishing points where the actual communication between tasks takes place. Based on these points, the end-to-end latency may be computed considering the basic paths of an effect chain within a hyperperiod of the communicating tasks. The analysis was then extended to consider task offsets. An offset assignment method was then suggested to further improve the determinism of the end-to-end latency, reducing control jitter. We finally showed the effectiveness of the LET model

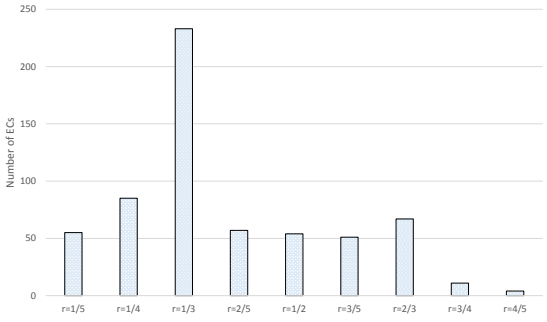


Figure 17: Heuristics vs. Brute force approach.

in achieving a more deterministic end-to-end communication delay for the effect chains of an industrial case study from the automotive domain. We presented a set of experiments showing that the jitter of the end-to-end latency with the LET model is in average within 1% for representative task sets, analytically confirming the control determinism of the LET model. However, non harmonic effect chains may have significantly higher jitters. In these cases, a considerable jitter reduction can be obtained using the proposed offset assignment heuristics.

As a future work, we intend to analytically and experimentally compare end-to-end age and reaction delays for the LET model against the implicit and explicit communication counterparts. While the LET model allows significantly reducing the variability in the end-to-end communication delays, the absolute latencies tend to be higher than those with other communication paradigms where tasks publish their computed result at an earlier time. We believe that a thorough comparing study is in order to understand pros and cons of each communication model, paving the way towards a generalized method that allows obtaining smaller end-to-end latencies within a reduced jitter.

REFERENCES

[1] Anonymous. 2017. Details omitted for double blind review process.
[2] N.C. Audsley. 2001. On priority assignment in fixed priority scheduling. *Inform. Process. Lett.* 79, 1 (2001), 39 – 44.
[3] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2016. Synthesizing Job-Level Dependencies for Automotive Multi-Rate Effect Chains. In *The 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*.
[4] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. 2017. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture* 80, Supplement C (2017), 104 – 113.
[5] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. 2009. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society.
[6] Joël Goossens. 2003. Scheduling of Offset Free Systems. *Real-Time Syst.* 24, 2 (March 2003), 239–258.
[7] Mathieu Grenier, Lionel Havet, and Nicolas Navet. 2008. Pushing the limits of CAN - scheduling frames with offsets provides a major performance boost. In *4th European Congress on Embedded Real Time Software (ERTS 2008)*. Toulouse, France.
[8] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, and Falk Wurst. 2017. Communication Centric Design in Complex Automotive Embedded Systems. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017) (Leibniz International Proceedings in Informatics (LIPIcs))*, Marko Bertogna (Ed.), Vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 10:1–10:20.
[9] A. Hamann, D. Ziegenbein, S. Kramer, and M. Lukasiewycz. 2017. 2017 Formals Methods and Timing Verification (FMTV) challenge. 1–1.
[10] T. A. Henzinger, B. Horowitz, and C. M. Kirsch. 2003. Giotto: a time-triggered language for embedded programming. *Proc. IEEE* 91, 1 (Jan 2003), 84–99.
[11] T. A. Henzinger, C. M. Kirsch, M. A. A. Sanvido, and W. Pree. 2003. From control models to real-time code using Giotto. *IEEE Control Systems* 23, 1 (Feb 2003), 50–64.
[12] C.M. Kirsch and A. Sokolova. 2012. The Logical Execution Time Paradigm. In *Advances in Real-Time Systems*. 103–120. /pubpdf/ARTS-chapter.pdf
[13] T. Kloda, B. d'Ausbourg, and L. Santinelli. 2016. EDF schedulability test for the E-TDL time-triggered framework. In *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*. 1–10.
[14] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. 2015. Real world automotive benchmarks for free. In *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*.
[15] Steffen Lampke, Simon Schliecker, Dirk Ziegenbein, and Arne Hamann. 2015. Resource-Aware Control-Model-Based Co-Engineering of Control Algorithms and Real-Time Systems. *SAE International Journal of Passenger Cars-Electronic and Electrical Systems* 8, 2015-01-0168, 106–114.
[16] C. L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* 20, 1 (Jan. 1973), 46–61.
[17] P. Marti, R. Villa, J. M. Fuertes, and G. Fohle. 2001. On real-time control tasks schedulability. In *2001 European Control Conference (ECC)*. 2227–2232.
[18] A. Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. 2012. Multisource Software on Multicore Automotive ECUs; Combining Runnable Sequencing With Task Scheduling. *IEEE Transactions on Industrial Electronics* 59, 10 (Oct 2012), 3934–3942.
[19] Mitra Nasri, Robert I. Davis, and year=2017 Björn B. Brandenburg. [n. d.]. FIFO with Offsets: High Schedulability with Low Overheads.
[20] J. C. Palencia and M. Gonzalez Harbour. 1998. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No.98CB36279)*. 26–37.
[21] J. C. Palencia, M. G. Harbour, J. J. GutiÃĺrrez, and J. M. Rivas. 2017. Response-Time Analysis in Hierarchically-Scheduled Time-Partitioned Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 7 (July 2017), 2017–2030.
[22] J. C. Palencia, M. G. Harbour, J. J. GutiÃĺrrez, and J. M. Rivas. 2017. Response-Time Analysis in Hierarchically-Scheduled Time-Partitioned Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 7 (July 2017), 2017–2030.
[23] O. Redell and M. Torngren. 2002. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In *Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*. 164–172.
[24] Ken Tindell. 2007. Adding Time-offsets to Schedulability Analysis.
[25] Rémy Wyss, Frédéric Boniol, Claire Pagetti, and Julien Forget. 2013. End-to-end Latency Computation in a Multi-periodic Design. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. ACM, New York, NY, USA, 1682–1687.