Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains in Safety Critical Embedded Systems

Luiz Maia, Gerhard Fohler

Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains in Safety Critical Embedded Systems

Luiz Maia, Gerhard Fohler
University of Kaiserslautern-Landau, Germany
{maianeto,fohler}@rptu.de

***Abstract*—The *Logical Execution Time* (LET) model has deterministic timing and data-flow properties, which simplify the computation of end-to-end latencies of multi-rate cause-effect chains. However, the LET model results in pessimistic end-to-end latencies since it abstracts the underlying platform and scheduling choices. In this paper, we propose a method to reduce end-to-end latencies of multi-rate cause-effect chains applying the LET model, by considering knowledge of the schedule in later design phases of safety critical embedded systems. Our method shortens and shifts the communication intervals of the LET model. If needed, e.g., for legacy reasons, our method can be applied to a subset of tasks only. We evaluate our work based on automotive benchmarks and synthetic task sets. We compare our results with previous work and the LET model. The experiments show significant reductions of maximum *reaction time* and *data age* values.**

***Index Terms*—Safety Critical Embedded Systems, Real-Time Systems, End-to-End Timing Analysis, LET**

I. Introduction

Designing safety critical applications in embedded systems, such as in AUTOSAR, requires complex analysis for temporal properties, such as end-to-end latencies. During early design phases, designers abstract system semantics, e.g., scheduling algorithms, in order to reduce the complexity. However, abstracting system semantics in this manner results in pessimistic end-to-end latencies [1].

A *cause-effect chain* (CEC) represents a sequence of tasks, which are executed to achieve a given functionality. A typical example is a sensor to actuator CEC, which consists of a task that reads the sensor (*cause*), a task that processes the read value, and a task that writes the output to an actuator (*effect*).

The analysis of whether or not the end-to-end (E2E) latency between the cause and the effect respects system's timing requirements is not trivial [2]. Especially when the CEC contains tasks with different periods and multiple data dependencies. The complexity of computing E2E latencies of multi-rate CECs increases even further when considering multi-core systems as tasks can be mapped to different cores and can execute in parallel [3].

Depending on the adopted communication model, the points in time when inter-task communications (accesses to shared resources) occur can be non-deterministic. They depend on when tasks start and finish their execution.

The Logical Execution Time (LET) model [4] emerged as a solution which significantly reduces timing analysis complexity of multi-rate CECs. By having fixed inter-task communication points that are independent from the actual task execution, the LET model brings timing and data-flow determinism to the analysis of multi-rate CECs. In LET, inter-task communication only occurs at the boundaries of the so-called *communication interval* [5], which is considered equal to the period interval of the task. As a result, the LET model helps abstracting from the actual system implementation (scheduling choices), and consequently reduces complexity of analysis, but at the cost of increased pessimism, i.e., larger E2E latency values.

In this paper we propose a method to reduce the pessimism present in the LET model by taking scheduling choices into consideration. The method shortens and shifts communication intervals based on a chosen scheduling algorithm. Therefore, our method is applied later in the design process (after scheduling choice). By analyzing a feasible schedule, it derives new boundaries for the communication intervals.

Our method is built on the ideas of adding phase to specific tasks [6] and shortening the communication interval equal to task's worst-case response time [7].

As design phases progress and a schedule has to be determined, our method can be applied during later design phases to optimize the E2E latencies of multi-rate CECs applying the LET model. Without losing the timing and data-flow determinism of LET, our method keeps tasks periodic and with well-defined communication points. If needed, it can be applied individually to selected tasks and/or CECs during later design phases, e.g., for legacy reasons. Results from evaluation based on an automotive benchmark presented by BOSCH [8], as well as synthetic task sets show a reduction of $\approx 65\%$ for the E2E latencies, on average.

Summary of contributions: Our method

- significantly reduces the E2E latencies of multi-rate CECs applying the LET model
- shortens and shifts tasks' communication intervals by taking scheduling decisions into consideration
- keeps tasks periodic and with well-defined inter-task communication points
- can be applied individually to selected tasks and/or CECs for legacy reasons

II. Related Work

The two most commonly considered latencies when analyzing a multi-rate CEC are: *reaction time* (*First to First* semantic) and *data age* (*Last to Last* semantic) [9]. The reaction time measures the *reactivity* of the system. It is the time interval between the occurrence of an external event until the *first* output based on that event. Data age measures the *freshness* of data in the CEC. It is the time interval between a data sampled (read) by the first task in the CEC until the *last* output (actuation) based on such data is produced by the last task in the CEC. Recently Günzel et al. [10] showed that the values for the maximum reaction time and maximum data age are equivalent.

In order to take data propagation delays into account, Klaus et al. [11] proposed an extension of the Real-Time Systems Compiler (RTSC), while Forget et al. [12] proposed a language to do a formal verification of E2E constraints at the model level. Becker et al. [13] proposed to use job-level dependency as a way to control data propagation and E2E latencies in multi-rate CECs. In [14], Dürr et al. introduced the concept of *job chains* and provided an analysis of the maximum reaction time and maximum data age. Schlatow et al. presented in [15] an analysis of the data age for periodic offset-synchronized tasks. In [16], Günzel et al. presented a timing analysis of asynchronized distributed CECs.

The LET model [4] was first introduced as part of the Giotto programming language in the context of time-triggered tasks. In [17], Biondi et al. presented a method to implement the LET model using additional dedicated tasks to realize the logical behavior of LET. In [18], Pazzaglia et al. used LET to enforce causality and determinism as a way to control accesses to shared memory and optimize the functional deployment on multi-core platforms.

In [19], Kloda et al. proposed to decouple the communication interval from the periods of tasks as an extension for the Timing Definition Language (TDL) [20], a successor of Giotto. However, Kloda et al. [19] did not present a method to formally compute the additional offsets or how to actually decouple the communication intervals.

Techniques have been proposed to compute the E2E latencies of multi-rate CECs applying the LET model. Becker et al. [21] presented a method to compute the maximum data age considering different communication models. Kordon and Tang [22] proposed a method to determine the maximum data age based on a task dependency graph. In [6], Martinez et al. presented a phase-aware LET analysis to improve the E2E latencies of multi-rate CECs. In [7], Bradatsch et al. proposed a method to reduce data age by setting the communication intervals equal to tasks' worst-case response time.

We build our work on top of the ideas proposed by Martinez et al. [6] and Bradatsch et al. [7]. By taking scheduling decisions into consideration, our method shortens and shifts communication intervals while keeping tasks periodic and with well-defined inter-task communication points.

III. System Model

We consider a multi-core system composed of identical cores and a task set Γ containing periodic and independent real-time tasks.

A. Tasks and Jobs

A task τ is a tuple $(C_\tau, T_\tau, D_\tau, \phi_\tau)$, where C_τ represents the worst-case execution time (WCET), T_τ is the period, D_τ is the deadline, and ϕ_τ is the phase. We assume tasks have implicit deadlines, i.e., deadline is equal to period. A *job* J represents an instance of τ, where $J(i)$ is the i^{th} instance of τ, $i \in \mathbb{N}^+$. $J(i)$ has a release time at $\phi_\tau + (i-1)T_\tau$ and an absolute deadline D_τ time units later. A schedule $\mathcal{S}$ specifies the execution behavior of all jobs of τ according to a scheduling policy. The *start time* of $J(i)$ according to $\mathcal{S}$ is $s^{\mathcal{S}}_{J(i)}$, while the *finishing time* is $f^{\mathcal{S}}_{J(i)}$. If the choice of a schedule is clear, we omit the index $\mathcal{S}$ for all definitions.

B. Communication Model

We assume communication between tasks happens through the use of shared resources and to be based on the LET model. Each task τ has a fixed and well defined communication interval L_τ (Figure 1). The inputs and outputs of τ are logically updated at the boundaries of L_τ. $begin(L_\tau)$ and $end(L_\tau)$ are relative points in time representing the boundaries of L_τ, i.e., $L_\tau = [begin(L_\tau), end(L_\tau)]$. $|L_\tau|$ represents the length of interval L_τ.

Each job J of τ has a communication interval L_J. The boundaries of L_J define when a job J logically receives (read) input from a shared resource, as well as when it logically transmits (write) output to a shared resource. At $begin(L_J)$, the logical *read-event* of J from a shared resource occurs. For instance, if $begin(L_\tau) = 0$, that means the logical read-event of each $J \in \tau$ happens during its release. At $end(L_J)$, the logical *write-event* of J to a shared resource occurs.

Figure 1 shows the communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $|L_\tau| = T_\tau$, i.e., $L_\tau = [0, T_\tau]$.

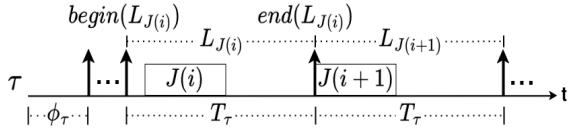

Fig. 1: Communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $L_\tau = [0, T_\tau]$

C. Cause-Effect Chain

A Cause-Effect Chain (CEC) represents an ordered sequence of communications carried out between a finite set of tasks. We represent a CEC by $E = (\tau_1 \rightarrow \tau_2 \rightarrow \cdots \rightarrow \tau_{|E|})$, $|E|$ being the number of tasks in E. The function $E(i)$ returns the i^{th} task in E, $i \in \{1, 2, \cdots, |E|\}$. The $\rightarrow$ operator indicates that τ_{i+1} acts as a consumer/reader task, while τ_i as a producer/writer task.

We assume that E samples (acquires data) at every $begin(J_1)$, J_1 being a job of task $E(1)$. We use z to represent the time interval between the occurrence of an external event (input) and its sampling by J_1. Likewise, z' represents the time interval between $end(J_{|E|})$ and the actuation (output).

D. Job Chain

Given a CEC E, a job chain $c^E = (J_1 \rightarrow \cdots \rightarrow J_{|E|})$ is a finite sequence of jobs representing one of the possible data propagation paths of E. In a job chain, the following requirements are respected:

- J_i is a job of $E(i)$, $i \in \{1, 2, \cdots, |E|\}$.
- The data written by J_i is read by J_{i+1}. That is, $end(L_{J_i}) \leq begin(L_{J_{i+1}})$ for all $i \in \{1, 2, \cdots, |E| - 1\}$

We use c_i^E to represent the i^{th} job chain of E, $i \in \mathbb{N}^+$, while function $l(c^E)$ returns the time interval between $end(L_{J_{|E|}})$ and $begin(L_{J_1})$.

IV. Manipulating LET Communication Intervals to Reduce End-to-End Latencies

As discussed in Section I, multi-rate CECs applying the LET model have timing and data-flow determinism, but pessimistic E2E latencies. By exploiting information from a feasible schedule, our method reduces the pessimism present in the LET model while maintaining its deterministic characteristics and the periodicity of tasks.

Instead of setting $|L_\tau| = T_\tau$, i.e., $L_\tau = [0, T_\tau]$ $\forall \tau \in \Gamma$, our method derives new relative points in time for $begin(L_\tau)$ and $end(L_\tau)$. By repositioning the boundaries of L_τ and therefore the boundaries of L_J, it can postpone the logical read-event of J and prepone the logical write-event of J.

A. Defining Schedule-Aware Intervals

In order to make L_τ schedule-aware, $\forall \tau \in \Gamma$, our method sets L_τ's length and position equal to a new time interval I_τ, where the length of I_τ is $C_\tau \leq |I_\tau| \leq T_\tau$. $begin(I_\tau)$ and $end(I_\tau)$ delimit the boundaries of I_τ, i.e., $I_\tau = [begin(I_\tau), end(I_\tau)]$.

The length and position of I_τ are defined according to schedule $\mathcal{S}$. As explained in Section III-A, $\mathcal{S}$ specifies the start time s_J and the finishing time f_J for all $J \in \tau$.

Below we define the terms *relative start time* (S_J) and *relative finishing time* (F_J) to derive the communication boundaries for I_τ.

Definition 1: **Relative Start Time (of a Job)**. Let $J(i)$ be the i^{th} job of task τ in schedule $\mathcal{S}$. The relative start time ($S_{J(i)}$) of a job is the start time of $J(i)$ minus its release time.

$$S_{J(i)} = s_{J(i)} - (\phi_\tau + (i-1)T_\tau) \tag{1}$$

Definition 2: **Relative Finishing Time (of a Job)**. Let $J(i)$ be the i^{th} job of task τ in schedule $\mathcal{S}$. The relative finishing time ($F_{J(i)}$) of a job is the finishing time of $J(i)$ minus its release time.

$$F_{J(i)} = f_{J(i)} - (\phi_\tau + (i-1)T_\tau) \tag{2}$$

Depending on when each J executes between its release and deadline, the values for S_J and F_J can change for each $J \in \tau$ (one job may execute early during its period, while another job may execute later). In order to keep the timing and data-flow determinism of LET when setting $L_\tau = I_\tau$, it is necessary to ensure that all $J \in \tau$ have a common periodic communication interval, i.e., respects S_J and F_J, for all $J \in \tau$.

Our method sets communication boundaries for I_τ by computing the *earliest relative start time* (ES_τ) and the *latest relative finishing time* (LF_τ) of a task τ based on $\mathcal{S}$.

Definition 3: **Earliest Relative Start Time (of a Task)**. Let τ be a task in schedule $\mathcal{S}$. The earliest relative start time (ES_τ) of τ is the minimum relative start time among all jobs of τ in $\mathcal{S}$.

$$ES_\tau = \min_{\forall J \in \tau} S_J \tag{3}$$

Definition 4: **Latest Relative Finishing Time (of a Task)**. Let τ be a task in schedule $\mathcal{S}$. The latest relative finishing time (LF_τ) of τ is the maximum relative finishing time among all jobs of τ in $\mathcal{S}$.

$$LF_\tau = \max_{\forall J \in \tau} F_J \tag{4}$$

In order to exemplify definitions 1 to 4, we show in Figure 2 a schedule $\mathcal{S}$ for three tasks that are part of a CEC E, $E = (\tau_1 \rightarrow \tau_2 \rightarrow \tau_3)$. In this example, we analyze task τ_2, which has three jobs in $\mathcal{S}$. Following definitions 1 and 2, the first job of τ_2, $J(1)$, has $S_{J(1)} = 2$ and $F_{J(1)} = 3$, while $J(2)$ has $S_{J(2)} = 0$ and $F_{J(2)} = 1$. $J(3)$ has $S_{J(3)} = 1$ and $F_{J(3)} = 2$. Following definitions 3 and 4, $ES_{\tau_2} = 0$ and $LF_{\tau_2} = 3$. By using definitions 1 to 4, ES_τ and LF_τ for the other tasks in E, $ES_{\tau_1} = 0$ and $LF_{\tau_1} = 1$ for τ_1, while $ES_{\tau_3} = 1$ and $LF_{\tau_3} = 2$ for τ_3.

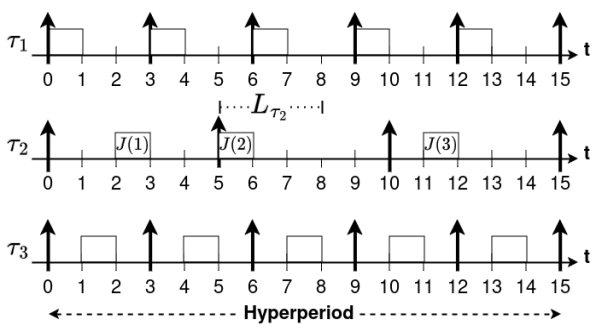

Fig. 2: Schedule $\mathcal{S}$ for a CEC $E = (\tau_1 \rightarrow \tau_2 \rightarrow \tau_3)$

Based on the values of ES_τ and LF_τ, our proposed method shifts and shortens the communication intervals of a given task τ. It repositions the communication intervals of τ by shifting them according to ES_τ. For instance, it sets the new phase of τ to be: $\phi_\tau = \phi'_\tau + ES_\tau$, ϕ'_τ is τ's initial phase. Our method shortens the length of τ's communication intervals according to LF_τ. Since the boundaries of I_τ are relative points in time with respect to τ and our proposed method shifted τ

according to ES_τ, $begin(I_\tau) = 0$ and $end(I_\tau) = LF_\tau - ES_\tau$. Therefore, by setting $I_\tau = [begin(I_\tau), end(I_\tau)]$, and $L_\tau = I_\tau$, our method shortens and shifts the communication intervals of τ. For instance, for task τ_2 shown in Figure 2, instead of setting $L_{\tau_2} = [0, T_{\tau_2}]$, i.e., $[0, 5]$, our method sets $L_{\tau_2} = I_{\tau_2} = [0, 3]$. Note that in this example τ_2 is not shifted because $ES_{\tau_2} = 0$.

In Figure 3, we show the communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $L_\tau = I_\tau$.

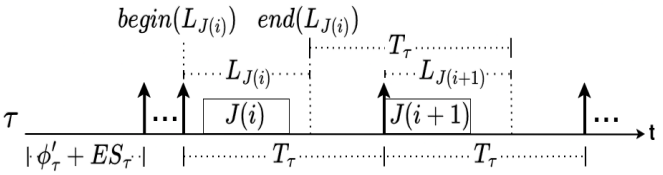

Fig. 3: Communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $L_\tau = I_\tau$

Note that although our method adds a phase ES_τ to a task τ, neither the schedulability of the task set nor jobs' execution order in schedule $\mathcal{S}$ are affected: for any $J(i) \in \tau$, $i \in \mathbb{N}^+$, there is no $J(i)$ that executes before $(i-1)T_\tau + ES_\tau$ according to $\mathcal{S}$. All J of τ have to wait at least ES_τ time units after its release in order to execute. Therefore, as long as our method postpones the release of τ by ES_τ time units, $\forall \tau \in \Gamma$, our method does not affect the schedulability of the task set and preserve the execution order of jobs in schedule $\mathcal{S}$.

B. Computing Communication Points

Since the logical read and write-events of J happen at well defined points in time, it is possible to identify the communication points where data propagates from one task to the other. Martinez et al. [6] presented an analysis to compute the communication points between two tasks applying the LET model assuming that $|L_\tau| = T_\tau$, $\forall \tau \in \Gamma$. As our method shortens and shifts communication intervals, the assumption does not hold anymore and the analysis is no longer applicable.

Inspired by the work done by Martinez et al. [6], we present a new analysis to compute the communication points of tasks applying the LET model assuming that $L_\tau = I_\tau$ in theorems 1 and 2. Below we define the terms *publishing point* and *reading point*, which are later used in theorems 1 and 2.

Definition 5: **Publishing Point**. Given a pair of tasks in a CEC E, where $\tau_i \rightarrow \tau_{i+1}$, $i \in \{1, 2, \cdots, |E| - 1\}$. Let a publishing point ($P^n_{\tau_i,\tau_{i+1}}$) be the n^{th} point in time where the resource shared by τ_i and τ_{i+1} is updated by τ_i. After $P^n_{\tau_i,\tau_{i+1}}$, no other logical write-event of τ_i will take place before the next logical read-event of τ_{i+1}.

Definition 6: **Reading Point**. Given a pair of tasks in a CEC E, where $\tau_i \rightarrow \tau_{i+1}$, $i \in \{1, 2, \cdots, |E| - 1\}$. Let a reading point ($Q^n_{\tau_i,\tau_{i+1}}$) be the n^{th} point in time where the resource shared by τ_i and τ_{i+1} is read by τ_{i+1}. The logical read-event of τ_{i+1} after the n^{th} publishing point of τ_i is the reading point $Q^n_{\tau_i,\tau_{i+1}}$.

Theorem 1: Let $\tau_i \rightarrow \tau_{i+1}$ be a pair of tasks applying the LET model with schedule-aware intervals in a CEC E, where $T_{\tau_i} \leq T_{\tau_{i+1}}$, $i \in \{1, 2, \cdots, |E| - 1\}$. Then the reading and publishing points between τ_i and τ_{i+1} can be computed as:

$$Q^n_{\tau_i,\tau_{i+1}} = nT_{\tau_{i+1}} + \phi_{\tau_{i+1}} \tag{5}$$

$$P^n_{\tau_i,\tau_{i+1}} = \left\lfloor \frac{Q^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_i} - end(L_{\tau_i})}{T_{\tau_i}} \right\rfloor T_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i}) \tag{6}$$

$$n \geq \begin{cases} 0, & \text{if } \phi_{\tau_{i+1}} \geq \phi_{\tau_i} + end(L_{\tau_i}) \\ \left\lceil \frac{\phi_{\tau_i} + end(L_{\tau_i}) - \phi_{\tau_{i+1}}}{T_{i+1}} \right\rceil, & \text{otherwise} \end{cases}$$

Proof: We prove this theorem in two steps.

Step 1 (Reading point): Since $T_{\tau_i} \leq T_{\tau_{i+1}}$, there is always one job of τ_i being released between two job releases of τ_{i+1}. That means, τ_i always updates the resource shared with τ_{i+1} before each logical read-event of τ_{i+1}. By definition (Section III-B), the inputs of τ_{i+1} are logically updated at $begin(L_{\tau_{i+1}})$, which occurs every $T_{\tau_{i+1}}$ time units after $\phi_{\tau_{i+1}}$. Therefore, a reading point between τ_i and τ_{i+1} occurs every $nT_{\tau_{i+1}} + \phi_{\tau_{i+1}}$.

Step 2 (Publishing point): By definition (Section III-B), the logical write-event of the *first* job of τ_i logically occurs at $\phi_{\tau_i} + end(L_{\tau_i})$. That means, any reading point $Q^n_{\tau_i,\tau_{i+1}}$ has to be $\geq$ than $\phi_{\tau_i} + end(L_{\tau_i})$. If $Q^n_{\tau_i,\tau_{i+1}} = \phi_{\tau_i} + end(L_{\tau_i})$, then $P^n_{\tau_i,\tau_{i+1}} = Q^n_{\tau_i,\tau_{i+1}}$. If $Q^n_{\tau_i,\tau_{i+1}} > \phi_{\tau_i} + end(L_{\tau_i})$, the publishing point that immediately precedes $Q^n_{\tau_i,\tau_{i+1}}$ depends on how many logical write-events of τ_i happened within the interval $[\phi_{\tau_i} + end(L_{\tau_i}), Q^n_{\tau_i,\tau_{i+1}}]$. Since logical write-events of τ_i occur periodically according to T_{τ_i}, the number of logical write-events within the considered interval is $\left\lfloor \frac{Q^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_i} - end(L_{\tau_i})}{T_{\tau_i}} \right\rfloor$. Hence, the publishing point that immediate precedes $Q^n_{\tau_i,\tau_{i+1}}$ is at $\left\lfloor \frac{Q^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_i} - end(L_{\tau_i})}{T_{\tau_i}} \right\rfloor T_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i})$. ■

Theorem 2: Let $\tau_i \rightarrow \tau_{i+1}$ be a pair of tasks applying the LET model with schedule-aware intervals in a CEC E, where $T_{\tau_i} > T_{\tau_{i+1}}$ and $i \in \{1, 2, \cdots, |E| - 1\}$. Then the publishing and reading points between τ_i and τ_{i+1} can be computed as:

$$P^n_{\tau_i,\tau_{i+1}} = nT_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i}) \tag{7}$$

$$Q^n_{\tau_i,\tau_{i+1}} = \left\lceil \frac{P^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil T_{\tau_{i+1}} + \phi_{\tau_{i+1}} \tag{8}$$

$$n \geq \begin{cases} 0, & \text{if } \phi_{\tau_{i+1}} \leq \phi_{\tau_i} + end(L_{\tau_i}) \\ \left\lfloor \frac{\phi_{\tau_{i+1}} - (\phi_{\tau_i} + end(L_{\tau_i}))}{T_i} \right\rfloor, & \text{otherwise} \end{cases}$$

Proof: We prove this theorem in two steps.

Step 1 (Publishing point): Since $T_{\tau_i} > T_{\tau_{i+1}}$, there is always one job of τ_{i+1} being released between two job releases of τ_i. That means, τ_{i+1} always reads the resource shared with τ_i after each logical write-event of τ_i. By definition (Section III-B), the outputs of τ_i are logically updated at $end(L_{\tau_i})$, which occurs

every T_{τ_i} after $\phi_{\tau_i} + end(L_{\tau_i})$. Therefore, a publishing point between τ_i and τ_{i+1} occurs every $nT_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i})$.

Step 2 (Reading point): By definition (Section III-B), the logical read-event of the *first* job of τ_{i+1} logically occurs at $begin(L_{\tau_{i+1}})$, i.e., $\phi_{\tau_{i+1}}$. That means, any reading point $Q^n_{\tau_i,\tau_{i+1}}$ has to be $\geq$ than $\phi_{\tau_{i+1}}$. By intuition, if $P^n_{\tau_i,\tau_{i+1}} \leq \phi_{\tau_{i+1}}$, then $Q^n_{\tau_i,\tau_{i+1}} = \phi_{\tau_{i+1}}$. If $P^n_{\tau_i,\tau_{i+1}} > \phi_{\tau_{i+1}}$, the reading point that immediately succeeds $P^n_{\tau_i,\tau_{i+1}}$ depends on how many logical read-events of τ_{i+1} happened within the interval $[\phi_{\tau_{i+1}}, P^n_{\tau_i,\tau_{i+1}}]$. Since logical read-events of τ_{i+1} occur periodically according to $T_{\tau_{i+1}}$, the number of logical read-events within the considered interval is $\left\lceil \frac{P^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil$. Hence, intuitively, the read point of $\tau_i \rightarrow \tau_{i+1}$ that immediately succeeds $P^n_{\tau_i,\tau_{i+1}}$ is at $\left\lceil \frac{P^n_{\tau_i,\tau_{i+1}} - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil T_{\tau_{i+1}} + \phi_{\tau_{i+1}}$. ■

In Section V, we show how to compute the E2E latencies of a given CEC using theorems 1 and 2.

V. Computing End-to-End Latencies

In Section IV we presented a method that exploits information from a schedule and define new communication intervals for tasks applying the LET model. Since our method shortens and shifts the communication intervals, we derived theorems 1 and 2 to identify the points in time when data propagates from one task to another in a CEC. In the following, we demonstrate how to identify which job chains have to be investigated during the E2E analysis of a multi-rate CEC. We also show how to compute the maximum reaction time and data age latencies related to those job chains.

A. Identifying Job Chains

Due to the under/oversampling nature of multi-rate CECs, some job chains might have jobs in common. For instance, when multiple actuations (outputs) of a CEC E are based on the same sampled (input) data, multiple job chains have the same $J(i)$ as their J_1 in E. As mentioned in Section II, when analyzing the data age of a multi-rate CEC, it is necessary to know for how long a sampled value affects the actuation of the CEC, i.e., know the time interval between two consecutive job chains with different $J(i)$ as their J_1. In order to distinguish job chains that have different $J(i)$ as their J_1, below we define the term *primary job chain*.

Definition 7: **Primary Job Chain**. Given a set of job chains that have the same $J(i)$ as their J_1. We call *primary job chain* (pc^E), the job chain with the *earliest* $end(L_{J_{|E|}})$ in the set. $pc^E_i = (J_1 \rightarrow J_2 \rightarrow \cdots \rightarrow J_{|E|})$ represents the i^{th} primary job chain of E, $i \in \mathbb{N}^+$. Given a prime job chain pc^E_i, we represent the next prime job chain of E after pc^E_i as pc^E_{i+1}. That is, pc^E_{i+1} is the first job chain after pc^E_i that has a different $J(i)$ as J_1. Function $pc^E_i(k)$ returns the communication interval $L_{J(k)}$ of the k^{th} job in the given primary job chain, $k \in \{1, 2, \cdots, |E|\}$. Function $l(pc^E_i)$ return the time interval between $end(pc^E_i(|E|))$ and $begin(pc^E_i(1))$.

In order to identify the primary job chains of a given CEC E, our method first identifies the job chains of E using algorithms 1 and 2. As shown by Leung et al. [23], for a task set with periodic tasks and offsets, a schedule $\mathcal{S}$ repeats itself every $LCM(T_{\tau_1}, \cdots, T_{\tau_{|E|}})$ units of time. The repetition starts at: $\Phi(E) + H(E)$, $\Phi(E) = \max(\phi_{\tau_1}, \cdots, \phi_{\tau_{|E|}})$ and $H(E) = LCM(T_{\tau_1}, \cdots, T_{\tau_{|E|}})$. Günzel et al. [10] showed that for the sake of computing the maximum reaction time and data age latencies of a given CEC E applying the LET model, it is enough to analyze the job chains within one of the repetition intervals after the *warm-up* period. The warm-up period covers the time interval required for an input to fully traverse E for the first time [10].

In order to identify which jobs are part of the job chains within one of the repetition intervals, our method applies Algorithm 1 to all jobs of $E(|E|)$ released within the repetition interval. The process of identifying jobs chains starts with the $J_{|E|}$ that has the *earliest* $begin(L_{J_{|E|}})$ within the repetition interval. Our method starts the analysis with the last communication task pair in E, i.e., $\tau_{|E|-1} \rightarrow \tau_{|E|}$. By applying Algorithm 1 to a given $J_{|E|}$ of $E(|E|)$, our method obtains the publishing point related to a $J_{|E|-1}$.

Algorithm 1 Compute publishing points

Input: $begin(L_{J_i}), \tau_{i-1}, \tau_i$

1: **if** $T_{\tau_{i-1}} \leq T_{\tau_i}$ **then**
2: According to Theorem 1
3: **else**
4: According to Theorem 2
5: **end if**
6: Find $m_{\max}$, the largest value of m such that $P^m_{\tau_{i-1},\tau_i} \leq begin(L_{J_i})$

Output: $P^m_{\tau_{i-1},\tau_i}$

Using the output of Algorithm 1 as an input to Algorithm 2, our method computes the publishing and reading points of the previous communication task pair in E, i.e., $\tau_{|E|-2} \rightarrow \tau_{|E|-1}$. Note that the process of identifying a job chain' starts from its tail ($J_{|E|}$) and stops when its head (J_1) is found. Our method identifies the remaining part of the job chain by applying Algorithm 2 recursively to all the other communication task pairs in E until our method obtains the reading and publishing point related to a job J_1 of $E(1)$.

Algorithm 2 Compute reading points

Input: $P^n_{\tau_{i-1},\tau_i}, \tau_{i-2}, \tau_{i-1}$

1: **if** $T_{\tau_{i-2}} \leq T_{\tau_{i-1}}$ **then**
2: According to Theorem 1
3: **else**
4: According to Theorem 2
5: **end if**
6: Find $m_{\max}$, the largest value of m such that $Q^m_{\tau_{i-2},\tau_{i-1}} < P^n_{\tau_{i-1},\tau_i}$
7: $n = m_{\max}$
8: Compute $Q^n_{\tau_{i-2},\tau_{i-1}}$ and $P^n_{\tau_{i-2},\tau_{i-1}}$
9: $i = i - 1$

Output: $Q^n_{\tau_{i-1},\tau_i}$ and $P^n_{\tau_{i-1},\tau_i}$

Our method uses Equation 9 on each identified publishing point ($P^n_{\tau_k,\tau_{k+1}}$), $k \in \{1, 2, \cdots, |E|-1\}$, in order to determine the i^{th} job of each task τ_k that is part of the communication task pairs in the CEC E. Note that reading points could also be used to determine the i^{th} job of each task τ_k, i.e., $i = 1 + \frac{Q^n_{\tau_k,\tau_{k+1}} - \phi_{\tau_k}}{T_{\tau_k}}$.

$$i = 1 + \frac{P^n_{\tau_k,\tau_{k+1}} - end(L_{\tau_k})}{T_{\tau_k}} \tag{9}$$

Following Definition 7, our method selects, within the repetition interval, the primary job chains among the set of job chains identified using algorithms 1 and 2. Our method adds the selected primary job chains to a set ζ and use it to compute the maximum reaction time and data age latencies.

B. Computing the Maximum Reaction Time and Data Age Latencies

As discussed in Section V-A, in order to compute de maximum reaction time and data age latencies of a given CEC E, our method first needs to identify all primary job chains of E within one of the repetition intervals after the warm-up period. In this section we show how to compute the E2E latencies of a CEC E given the set of primary job chains within a repetition interval.

We follow the definitions of maximum reaction time and data age latencies used by Günzel et al. [10]. As mentioned in Section III, we consider an additional delay z between the occurrence of an external event (input) and its sampling by J_1. In the same manner, we consider that z' represents an additional delay between $end(L_{J_{|E|}})$ and the actuation (output). Therefore, in order to compute the maximum reaction time and data age latencies as done by Günzel et al. [10], we append z to the beginning of a primary job chain pc^E and z' to its end, i.e., $pc^E = (z, J_1, \cdots, J_{|E|}, z')$. Despite the fact that the maximum latency values of the reaction time and data age are equivalent [10], in the following we also demonstrate how to calculate individually the intermediate values of those two latency metrics.

1) ***Maximum Reaction Time****:* For a given CEC E and its set of primary job chains ζ, our method computes the reaction time for all $pc^E_i \in \zeta$ considering the maximum delay z that an incoming input could suffer. In the worst case, for a primary job chain pc^E_i, an input arrives right *after* the logical read-event of the first job of pc^E_i. As a result, the incoming input has to wait until the logical read-event of the next primary job chain (pc^E_{i+1}) in order to be recognized and propagated through E. We consider, as Günzel et al. [10], that actuation takes place at the logical write-event of $pc^E_{i+1}(|E|)$, i.e., $z' = 0$.

Equation 10 shows how to compute the reaction time of a given primary job chain pc^E_i assuming the maximum input delay.

$$RT(pc^E_i) = z + l(pc^E_{i+1}) + z' \tag{10}$$

where:

$z = begin(pc^E_{i+1}(1)) - begin(pc^E_i(1))$ and $z' = 0$

The maximum reaction time of a CEC E is:

$$MRT(E) = \max_{\forall pc^E_i \in \zeta} RT(pc^E_i) \tag{11}$$

2) ***Maximum Data Age****:* For a given CEC E and its set of primary job chains ζ, our method computes the data age for all $pc^E_i \in \zeta$ considering the maximum delay z' that an actuation output could suffer. In the worst case, for a primary job chain pc^E_i, the last actuation based on a given input happens right *before* the logical write-event of the last job of pc^E_{i+1}. We consider, as Günzel et al. [10], that the input data on which the outputs are based is read during the logical read-event of $pc^E_i(1)$, i.e., $z = 0$.

Equation 12 shows how to compute the data age of a given primary job chain pc^E_i assuming the maximum output delay.

$$DA(pc^E_i) = z + l(pc^E_i) + z' \tag{12}$$

where: $z = 0$ and $z' = end(pc^E_{i+1}(|E|)) - end(pc^E_i(|E|))$

The maximum data age of a CEC E is:

$$MDA(E) = \max_{\forall pc^E_i \in \zeta} DA(pc^E_i) \tag{13}$$

VI. Practically Enforcing Deterministic Communication Points

The LET model assumes that tasks' inputs and outputs are logically updated at the beginning and end of their communication intervals. However, platforms are not infinitely fast to realize such behavior. Therefore, the deterministic ordering and atomic execution of these updates must be enforced to preserve the desired logical behavior. At the implementation level, the LET model is enforced by using hardware/software mechanisms [24] [17]. In the literature, different ways to implement the logical behavior of the LET model have been proposed [2] [5] [17] [18] [25].

One way to implement the logical behavior of the LET model is by implementing auxiliary tasks, which are responsible for updating the inputs and outputs of the tasks [2] [17]. For instance, the implementation of a task τ would consist of three tasks: (i) a reader (*copy-in*) task τ^R; (ii) an execution task τ^E; (iii) a writer (*copy-out*) task τ^W. Hereafter, we discuss one of the possibilities of implementing the logical behavior of the LET model when shortening and shifting communication intervals. Note that other implementations respecting the logical behavior and the deterministic ordering of execution of the tasks could be used.

At the beginning of L_τ, the auxiliary reader task (τ^R) copies all the input data necessary for τ^E's execution to a local variable. At the end of L_τ, the auxiliary writer task (τ^W) copies τ^E's output data to the shared variable that will be accessed by the next task in the CEC. Therefore, tasks τ^R and τ^W are responsible for updating the inputs and outputs of τ at the boundaries of L_τ.

Since the auxiliary reader (resp. writer) task has to be executed as close as possible to the start (resp. end) of L_τ, τ^R and τ^W have to be characterized by a very short WCET and a

very high priority level [17]. Therefore, the correct positioning of τ^R and τ^W is important in order to achieve the expected logical behavior of τ [2].

We use the parameters of τ and its communication interval L_τ to set the parameters of τ^R, τ^E and τ^W. The auxiliary tasks τ^R and τ^W are periodic task with period T_{τ^R} (resp. T_{τ^W}) equal to T_τ. The correct positioning of τ^R and τ^W is done by means of an additional phase. Therefore, τ^R has a phase $\phi_{\tau^R} = \phi_{\tau^E} = \phi'_\tau + ES_\tau$, while τ^W has a phase $\phi_{\tau^W} = \phi'_\tau + LF_\tau$. We consider that the execution times of the copy operations done by τ^R and τ^W are very short and their overhead included in C_τ. The parameters $(C_{\tau^E}, T_{\tau^E}, D_{\tau^E})$ of τ^E are equal to the parameters of τ.

Figure 4 shows how the auxiliary tasks τ^R and τ^W of a given task τ can be modeled. Note that in Figure 4 $\tau^E = \tau$.

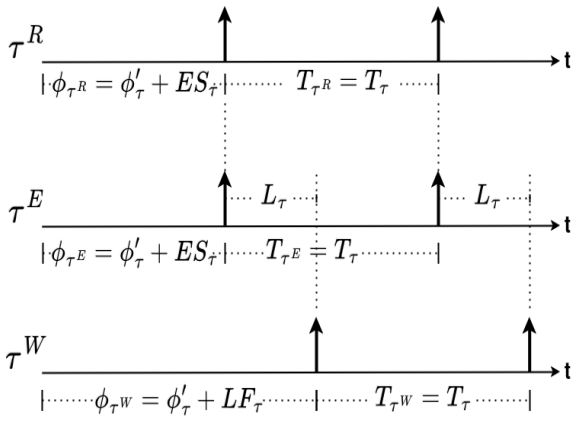

Fig. 4: Enforcing deterministic communication points by means of auxiliary tasks

The copying operations carried out by the auxiliary tasks (τ^R and τ^W) are facilitated via highest priority interrupts, which are added for each periodic auxiliary task. These interrupts are activated with the same period as the tasks they correspond to, allowing them to execute immediately. In order to obtain *freshest* data values, write output operations are favored over read input operations. Therefore, the write output operations are given the highest priority in the system, whereas read input operations are given the second highest priority.

VII. Experimental Results

We evaluate our work based on the Real World Automotive Benchmarks presented by BOSCH [8] and synthetic task sets. We compare our method with the approach presented by Bradatsch et al. [7] and the LET model. The method proposed by Bradatsch et al. [7] will be referenced as the WCRT-LET model. We consider that the task sets run on a system comprising four cores and that tasks are scheduled according to rate monotonic.

A. Real World Automotive Benchmarks

We generated and tested 500 schedulable task sets based on the parameters of the Real World Automotive Benchmarks [8]. We assign periods to tasks following the definitions of Table III in [8]. The range of possible periods is: [1, 2, 5, 10, 20, 50, 100, 200, 1000]ms. Note that the sum of the probabilities for possible periods in [8] is 85%. The remaining 15% is for angle-asynchronous tasks. Since, we do not consider angle-asynchronous tasks, we divided all probability values by 0.85. Inter-task communications follow the definitions of Table II in [8]. For each task we generated a WCET following the definitions of Tables IV and V in [8].

As stated in [8], in typical engine control applications there are between 30 and 60 CECs. In our experiments, on average, there are 38 CECs per task set. The number of periods per CEC is randomly chosen between the interval [1,3] following the definitions of Table VI in [8]. For each period that composes the CEC, there are 2 to 5 tasks with that same period (Table VII in [8]). Each CEC is composed of 2 to 15 tasks. The total utilization of cores is $\approx 71\%$ (per core), on average.

In figures 5 and 6, we show the maximum reaction time and data age results respectively. The box plots show the 25th percentile, average, 75th percentile and the maximum & minimum-case values. Note that in Figure 5, we normalized the results with respect to the maximum reaction time obtained by the LET model, while in Figure 6 we normalized the results with respect to the maximum data age.

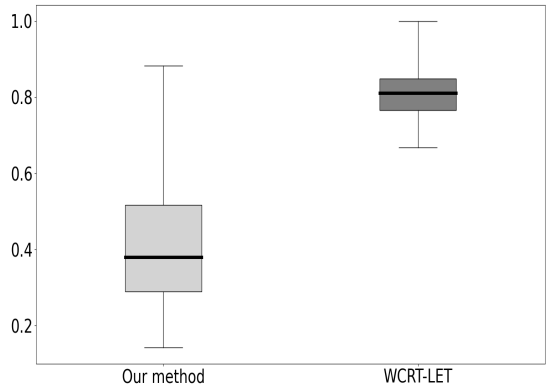

Fig. 5: Normalized maximum reaction time w.r.t LET

As shown in Figure 5, our model managed to obtain maximum reaction time values that are on average, $\approx 63\%$ lower than the values obtained by LET. Similarly, the WCRT-LET model managed to improve the maximum reaction time values by $\approx 20\%$.

As recently shown by Günzel et al. [10], values for the maximum reaction time and data age are equivalent. Figure 6 shows the results for the maximum data age. On average, our method obtained maximum data age values that are $\approx 63\%$ lower than the values obtained by LET. The WCRT-LET model improved the maximum reaction time values by $\approx 20\%$.

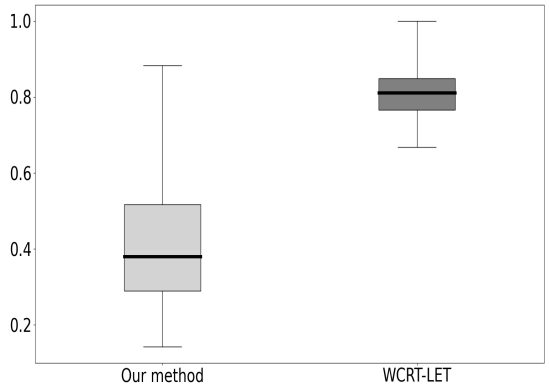

Fig. 6: Normalized maximum data age w.r.t LET

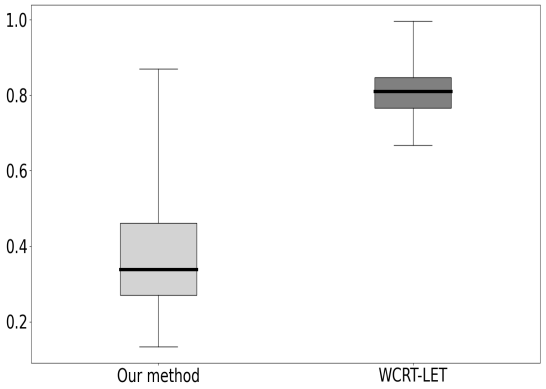

Fig. 7: Normalized maximum reaction time w.r.t LET

Table VI in [8] shows that 70% of the CECs present in the benchmarks are single-rate. Since our method shortens and shifts the communication intervals of the tasks, it makes possible for incoming inputs to propagate through the CEC within one repetition interval rather than in multiple as with $|L_i| = T_i$. As a result, in some cases our method resulted in an improvement exceeding 80% of that achieved using the LET model as shown in figures 5 and 6.

B. Synthetically Generated Workloads

We randomly generated 500 schedulable task sets, where we chose task periods and inter-task communication as in the previous experiment. However, this time we allowed tasks to have higher WCET values, increased the number of possible periods per CEC from 3 to 5 and reduced the probability of single-rate CECs from 70% to 7%. The 63% difference was split equally among the other possible number of periods. The new probabilities for possible number of periods per CEC are $\{1$: 7%, 2: 35.75%, 3: 25.75%, 4: 15.75%, 5: $15.75\%\}$.

As a result of increasing the number of possible periods in a CEC, we increased the total number of tasks composing it from 15 to 25. We chose randomly the amount of CECs per task set from interval $[10, 20]$, with an average of 13 CECs per task set. The total utilization of cores is $\approx 80\%$. The obtained results are summarized in figures 7 and 8.

As shown in figure 7 and 8, our method outperformed WCRT-LET and the LET model once again. For synthetic task sets, our model managed to obtain maximum reaction time values that are on average, $\approx 67\%$ lower than the values obtained by LET, while the WCRT-LET model managed to improve the maximum reaction time values by $\approx 20\%$.

In Figure 9, we analyze a single task set to further show how much improvement our method achieves over the LET model with respect to the maximum data age. We randomly selected one task set from the 500 schedulable task sets to analyze. The selected task set has 12 CECs and a total of 107 tasks distributed on the 4 cores. For example, the CEC with ID=9 has 18 tasks and 5 periods [20, 50, 100, 200, 1000]ms. Figure 9 shows that for the CEC with ID=9, our method improved the maximum data age value by $\approx 76\%$, while the WCRT-LET model improved it by $\approx 20\%$.

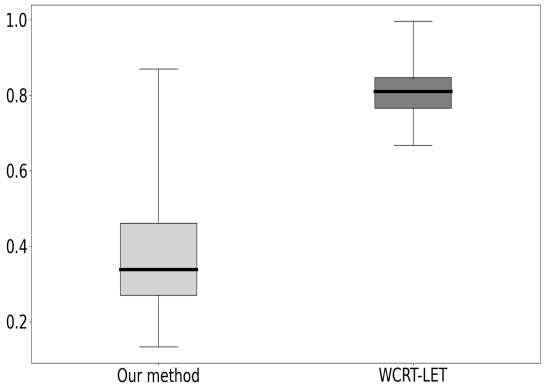

Fig. 8: Normalized maximum data age w.r.t LET

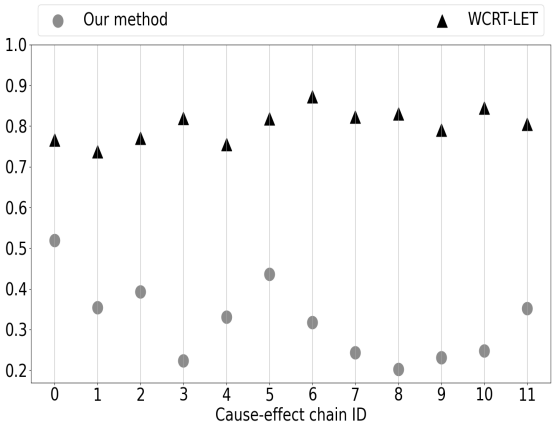

Fig. 9: Comparison of the normalized maximum data age values of the CECs present in a randomly selected task set

VIII. Conclusion

In this paper, we proposed a method to obtain less pessimistic end-to-end latencies of multi-rate cause-effect chains applying the LET model, by considering knowledge of the schedule in later design phases of safety critical applications.

Our method shortens and shifts communication intervals by exploiting schedule information, while maintaining the deterministic characteristics of the LET model and tasks' periodicity.

Experiments showed that for task sets based on the Real World Automotive Benchmarks presented by BOSCH [8] or randomly generated, our method produced less pessimistic end-to-end latencies than previous works. On both test case scenarios, it obtained end-to-end latency values that are, on average, $\approx 65\%$ lower when compared to the LET model.

If needed, e.g., for legacy reasons, our method does not have to be applied to the entire task set, but also to only a subset.

In future work, we plan to investigate the impact of our method in cause-effect chains containing different execution models and how it can influence the end-to-end latencies.

Acknowledgments

We would like to thank Denis Claraz, Vitesco Technologies France S.A.S, and the anonymous reviewers for their feedback and thorough comments. We are also grateful for all the feedback provided by the members of the Real Time Systems Chair at the University of Kaiserslautern-Landau.

References

[1] S. Matic and T. A. Henzinger, "Trading end-to-end latency for composability," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE, 2005, pp. 12–pp.

[2] 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 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[3] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 226–238.

[4] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *International Workshop on Embedded Software*. Springer, 2001, pp. 166–184.

[5] J. Martinez, I. Sanudo, P. Burgio, and M. Bertogna, "End-to-end latency characterization of implicit and let communication models," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), Dubrovnik, Croatia*, 2017.

[6] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.

[7] C. Bradatsch, F. Kluge, and T. Ungerer, "Data age diminution in the logical execution time model," in *International conference on Architecture of computing systems*. Springer, 2016, pp. 173–184.

[8] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, vol. 130, 2015.

[9] 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 *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.

[10] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, "On the equivalence of maximum reaction time and maximum data age for cause-effect chains," in *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.

[11] T. Klaus, F. Franzmann, M. Becker, and P. Ulbrich, "Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, 2018, pp. 93–103.

[12] 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)*. IEEE, 2017, pp. 1–8.

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

[14] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–24, 2019.

[15] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-age analysis and optimisation for cause-effect chains in automotive control systems," in *2018 IEEE 13th international symposium on industrial embedded systems (SIES)*. IEEE, 2018, pp. 1–9.

[16] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 40–52.

[17] A. Biondi, P. Pazzaglia, A. Balsini, and M. Di Natale, "Logical execution time implementation and memory optimization issues in autosar applications for multicores," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.

[18] P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 207–219.

[19] T. Kloda, B. d'Ausbourg, and L. Santinelli, "Towards a more flexible timing definition language," in *12th International Workshop Quantitative Aspects of Programming Languages and Systems-at ETAPS 2014*, 2014.

[20] W. Pree, J. Templ, P. Hintenaus, A. Naderlinger, and J. Pletzer, "Tdl-steps beyond giotto: A case for automated software construction." *Int. J. Softw. Informatics*, vol. 5, no. 1-2, pp. 335–354, 2011.

[21] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[22] A. Kordon and N. Tang, "Evaluation of the age latency of a real-time communicating system using the let paradigm," in *ECRTS 2020*, vol. 165. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020.

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

[24] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, pp. 1–27, 2021.

[25] A. Biondi and M. Di Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 240–250.