Bounds on Latencies and Robustness of Cause-Effect Chains in Automotive Real-Time Systems

Leonie Köhler
Institute for Computer and Network Engineering, TU Braunschweig, Germany

Max-Jonas Friese
Daimler AG, Stuttgart, Germany

Simon Bagschik
Institute for Computer and Network Engineering, TU Braunschweig, Germany

Rolf Ernst
Institute for Computer and Network Engineering, TU Braunschweig, Germany

─── **Abstract** ───────────────────────────────

In automotive real-time systems, the primary timing constraints relate to cause-effect chains in the application software. A cause-effect chain describes the entire process of reading sensor data, computing a control algorithm, driving actuators, and must satisfy a given end-to-end deadline. Therefore, it is not sufficient to compute response times of isolated tasks to verify correct timing, but also the end-to-end latency of the cause-effect chains must be derived. This paper extend existing approaches for the computation of end-to-end latencies to cover realistic systems models with distributed, mixed-triggered cause-effect chains. This paper also addresses a often neglected but very important aspect in industrial practice, namely the robustness of timing properties in the presence of software updates. Specifically, we derive robustness margins which guarantee that if software extensions stay within this bounds, then the end-to-end deadlines of cause-effect chains can still be satisfied.

1 Introduction

A timing-critical cause-effect chain typically includes the following steps: reading and processing of sensor signals, computing a control law, and control of actuators. Given the distributed nature of sensors, ECUs, and actuators in a car, a cause-effect chain in an automotive real-time system is typically distributed over several system components and requires off-chip communication via data busses and/or networks. There are two major problems related to the topic of timing-critical cause-effect chain, which are also addressed in this paper:

1. Computing an upper bound on the end-to-end latency of a distributed cause-effect chain. The performance analysis of timing-critical cause-effect chains (hereafter: CEC-analysis) has been topic of a number of publications. It builds on top of the classical scheduling analysis, which computes worst-case response times for individual tasks. The focus of the CEC-analysis lies on the computation of the maximum end-to-end latency on the basis of results from the classical scheduling theory. To the best knowledge of the authors, existing CEC-analysis is either limited to the scope of cause-effect chains on a single core, a

data bus, a network etc., and/or a specific type of cause-effect chain, which is composed of exclusively time-triggered, logical execution time (LET)-triggered, or event-triggered tasks as will be detailed in the section on related work. The first goal of this paper is therefore to bring together the separate results in the CEC-analysis together to compute end-to-end latency for complex distributed cause-effect chains.

2. Bounding the impact of software updates on the maximum end-to-end latency of cause-effect chains. The second goal is to explore the robustness of time-triggered cause-effect chains. In industrial practice, application software is not constantly re-written but evolves from an approved and tested prior version which is known to satisfy the timing requirements. A very important problem, equally important to the problem of computing maximum end-to-end latencies of cause-effect chains, is to evaluate how robust a given software design with a set of timing-critical cause-effect chains towards updates. Ideally the software should be extensible without violating given end-to-end deadlines of cause-effect chains. The problem of quantifying the robustness of distributed cause-effect chains towards updates has not been treated so far to the best knowledge of the authors. This paper proposes a robustness metric and specifies a rule how to compute it for a given software design with time-triggered cause-effect chains.

This paper is structured as follows. To begin with, we discuss related work and introduce our system model. Then we address the first issue and develop a CEC-analysis providing an upper bound on the end-to-end latency of a complex distributed cause-effect chain. Thereafter the robustness of cause-effect chains is explored. Finally, we apply the presented theory to an industrial case study.

2   Related Work

The related work on computing a maximum end-to-end latency for a given cause-effect chain can be classified according to (1) the types of cause-effect chains covered, and (2) the (non-)distribution of the cause-effect chain over several elements of the execution platform, as shown in Table 1.

The existing work distinguishes four different types of cause-effect chains

- *time-triggered cause-effect chain* where each task is periodically activated with(out) activation/release offset,
- *LET-triggered cause-effect chains* where each task is periodically activated with(out) activation/release offset and has a time-deterministic read and write behavior,
- *sporadically-triggered cause-effect chain* where the activation pattern of each task is specified by a minimum and maximum inter-arrival time,
- *event-triggered cause-effect chain* where the header task is activated by an arbitrary activation pattern while all other tasks are exclusively activated by the completion event of the predecessor event in the chain.

but does not provide methods how to compute end-to-end latencies of a cause-effect chain which is built from different types. Our paper fill this gap.

While design space exploration for the improvement of a given (set of) cause-effect chains has been recently treated, e.g., by [18] [3], the extensibility or robustness of a already designed cause-effect chain has not been treated so far to the best knowledge of the authors. However, robustness is a very important aspect in industrial practice where changes to a design should be incremental and a complete re-design with each software update should be avoided.

	Non-distributed	Distributed
Time-triggered	Forget et al. 2010 [9][8]	Pagetti et al. 2011 [15]
	Becker et al. 2016 [1]	Puffitsch et al. [16]
	Becker et al. 2017 [2]	
LET-triggered	Becker et al. 2017 [2]	
	Martinez et al. 2018 [14]	
Sporadically-triggered		Dürr et al. 2019 [5]
Event-triggered	Schlatow et al. 2016 [17]	CPA [11]
	Girault et al. 2018 [10]	Network calculus [13]
		and others
Mixed-triggered	Becker et al. 2017 [2]	

Table 1 Related work on computing the maximum end-to-end latency of cause-effect chains.

3 System Model

A *hardware platform* $\mathcal{P} = (\mathcal{R}, \mathcal{E})$ is a directed graph. A vertex of the graph is an element from the set of resources $\mathcal{R}$. The edges $\mathcal{E}$ correspond to the directed physical connections between resources. A *resource* $R \in \mathcal{R}$ is characterized by the property that it provides processing or communication service to tasks according to a given scheduling policy. A resource can thus be a processor core, a data bus, etc. In this paper, we assume that the clocks of all resources are perfectly synchronized.

A *task* $\tau_i \in \mathcal{T}$ is a service-consuming entity. This definition is very general and corresponds to the concept of a task in the real-time community. Since consumed service can be processing service or communication service; we do not need to distinguish between notions like "software tasks" and "messages". The jth instance of a task τ_i is called job $\tau_i(j)$. In the context of this paper, two variants of a task exist:

- the ***bounded execution time (BET) task*** characterized by the following attributes
  - *Arbitrary activation model.* Let $\underline{\delta}_i(n)$, resp. $\overline{\delta}_i(n)$, denote the minimum, resp. maximum, distance of n consecutive activation events of task τ_i.
  - *Release offset.* A constant release offset Φ_{τ_i} applies after each activation of task τ_i.
  - *Execution time.* The amount of execution time that a job $\tau_i(j)$ may require is bounded from above by the worst-case execution time $WCET_{\tau_i}$.
  - *Response time.* The response time of a job $\tau_i(j)$ is the duration between its release and completion. It is bounded by the best-case response time $BCRT_{\tau_i}$ and the worst-case response time $WCRT_{\tau_i}$.
  - *Relative deadline.* The relative deadline d_{τ_i} is added to the release instant of a job $\tau_i(j)$ to determine the latest acceptable moment of completion of job $\tau_i(j)$.
  - *Read-execute-write semantics.* A task with read-process-write semantics reads all required inputs before it starts to process. All outputs are written to registers directly after the data processing is finished.
  - *Allocation.* A task is associated with the resource from which it receives service.
  - **Note:** All existing methods for end-to-end latency analysis build on top of conventional worst-case response time analysis for tasks. Also in this paper, we assume that the worst-case response times (WCRTs) as well as activation models of tasks have already been computed before hand.
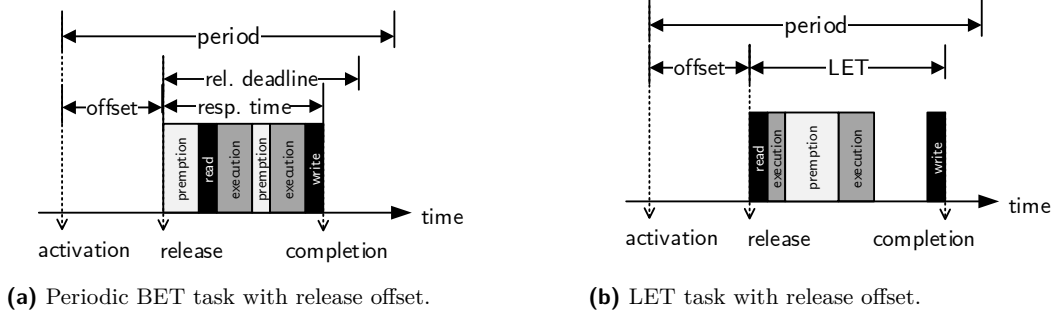- the ***logical execution time (LET) task*** characterized by

(a) Periodic BET task with release offset.

(b) LET task with release offset.

Figure 1 Task model.

- *Periodic activation model.* A task τ_i is periodically activated with period P_{τ_i}.
- *Release offset.* A constant release offset Φ_{τ_i} applies after each activation of task τ_i.
- *Logical execution time.* A task with logical execution time (LET) semantics reads all required inputs at its release time. All outputs are written to registers with the elapse of the LET_{τ_i}.
- *Allocation.* A task is associated with the resource from which it receives service.

An *application* is a directed graph $\mathcal{A} = (\mathcal{T}, \mathcal{D})$, where the set of vertices $\mathcal{T}$ represents the tasks in the application and the set of directed edges $\mathcal{D}$ represents the data-flow dependencies among the tasks.

▶ **Definition 1** (Cause-effect chain)**.** *A cause-effect chain $C = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$ is a finite directed walk in an application $\mathcal{A}$ which is executed on a hardware platform $\mathcal{P}$.*

▶ **Definition 2** (Instance of a cause-effect chain)**.** *An instance of a cause-effect chain $C(j_1, j_2, \ldots, j_n)$ is a sequence of jobs $(\tau_1^c(j_1), \tau_2^c(j_2), \ldots, \tau_n^c(j_n))$, where each job $\tau_{k+1}^c(j_{k+1})$ reads data from the predecessor job $\tau_k^c(j_k)$.*

The concepts of a cause-effect chain and an instance of a cause-effect chain are illustrated in Figure 2. The possible instances of a cause-effect chain can be determined by a data flow analysis, which checks whether a data flow through the sequence of jobs $\tau_1^c(j_1)\tau_2^c(j_2), \ldots, \tau_n^c(j_n)$ is at all possible.
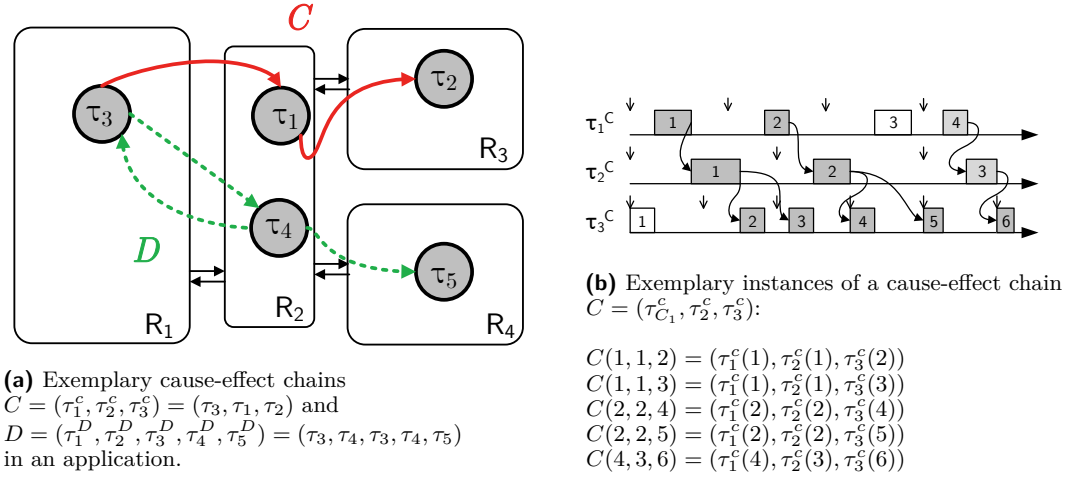
4 Data Flow Analysis: Computing Instances of a Cause-Effect Chain

The objective of this section is to present a data flow analysis, which delivers all possible instances of a given cause-effect chain. Becker et al. [2] introduced a data flow analysis for a single-resource system with a fully periodic task set. The fundamental principle of this data flow analysis, however, is valid for our more general system model with multiple resources and tasks with an arbitrary activation model as shown in Theorem 5.

▶ **Definition 3** (Read interval)**.** *The read interval of a job $\tau_i^c(j_i)$, denoted as*

$$RI\left(\tau_i^c(j_i)\right) = [\underline{R}\left(\tau_i^c(j_i)\right), \overline{R}\left(\tau_i^c(j_i)\right)],$$

is the time interval from the earliest possible instant to the latest possible instant at which job $\tau_i^c(j_i)$ can read its input data without violating its deadline or LET.

(a) Exemplary cause-effect chains
$C = (\tau_1^c, \tau_2^c, \tau_3^c) = (\tau_3, \tau_1, \tau_2)$ and
$D = (\tau_1^D, \tau_2^D, \tau_3^D, \tau_4^D, \tau_5^D) = (\tau_3, \tau_4, \tau_3, \tau_4, \tau_5)$
in an application.

(b) Exemplary instances of a cause-effect chain
$C = (\tau_{C_1}^c, \tau_2^c, \tau_3^c)$:

$C(1, 1, 2) = (\tau_1^c(1), \tau_2^c(1), \tau_3^c(2))$
$C(1, 1, 3) = (\tau_1^c(1), \tau_2^c(1), \tau_3^c(3))$
$C(2, 2, 4) = (\tau_1^c(2), \tau_2^c(2), \tau_3^c(4))$
$C(2, 2, 5) = (\tau_1^c(2), \tau_2^c(2), \tau_3^c(5))$
$C(4, 3, 6) = (\tau_1^c(4), \tau_2^c(3), \tau_3^c(6))$

■ **Figure 2** Modeling cause-effect chains.

▶ **Definition 4** (Data interval). *The data interval of a job $\tau_{C_i}^c(j_i)$, denoted as*

$$DI\left(\tau_i^c(j_i)\right) = \left[\underline{D}\left(\tau_i^c(j_i)\right), \overline{D}\left(\tau_i^c(j_i)\right)\right],$$

is the time interval from the earliest possible instant to the latest possible instant at which output data of job $\tau_i^c(j_i)$ is available without violating its deadline or LET.

▶ **Theorem 5** (Data flow analysis for a general cause-effect chain). *Assume an application $\mathcal{A}$ which is executed on a hardware platform as described in Section 3. Let the application $\mathcal{A}$ consists only of schedulable tasks, i.e., each task satisfies its deadline or LET. A finite directed walk in application $\mathcal{A}$ is the cause-effect chain C.*
Then all possible instances of a cause-effect chain C are contained in the (super)set

$$\mathcal{C} = \{C(j_1, j_2, \ldots, j_n) \mid j_1, j_2, \ldots, j_n \in \mathbb{N} \,\wedge$$
$$\forall k \in \mathbb{N} : 1 \leq k \leq n - 1 : RI\left(\tau_{k+1}^c(j_{k+1})\right) \cap DI\left(\tau_k^c(j_k)\right) \neq \emptyset\}$$

where

$$RI\left(\tau_i^c(j_i)\right) = \left[\underline{R}\left(\tau_i^c(j_i)\right), \overline{R}\left(\tau_i^c(j_i)\right)\right]$$

$$\underline{R}\left(\tau_i^c(j_i)\right) = \begin{cases} \underline{\delta}_{\tau_i^c}(j_i) + \Phi_{\tau_i^c} & \text{if } \tau_i^c \text{ is a BET task} \\ (j_i - 1) \cdot P_{\tau_i^c} + \Phi_{\tau_i^c} & \text{if } \tau_i^c \text{ is a LET task} \end{cases}$$

$$\overline{R}\left(\tau_i^c(j_i)\right) = \begin{cases} \overline{\delta}_{\tau_i^c}(j_i) + \Phi_{\tau_i^c} + d_{\tau_i^c} - BCRT_{\tau_i^c} & \text{if } \tau_i^c \text{ is a BET task} \\ (j_i - 1) \cdot P_{\tau_i^c} + \Phi_{\tau_i^c} & \text{if } \tau_i^c \text{ is a LET task} \end{cases}$$

and

$$DI\left(\tau_i^c(j_i)\right) = \left[\underline{D}\left(\tau_i^c(j_i)\right), \overline{D}\left(\tau_i^c(j_i)\right)\right]$$

$$\underline{D}\left(\tau_i^c(j_i)\right) = \begin{cases} \underline{\delta}_{\tau_i^c}(j_i) + \Phi_{\tau_i^c} + BCRT_{\tau_i^c} & \text{if } \tau_i^c \text{ is a BET task} \\ (j_i - 1) \cdot P_{\tau_i^c} + \Phi_{\tau_i^c} + LET_{\tau_i^c} & \text{if } \tau_i^c \text{ is a LET task} \end{cases}$$

$$\overline{D}\left(\tau_i^c(j_i)\right) = \begin{cases} \overline{\delta}_{\tau_i^c}(j_i + 1) + \Phi_{\tau_i^c} + WCRT_{\tau_i^c} & \text{if } \tau_i^c \text{ is a BET task} \\ j_i \cdot P_{\tau_i^c} + \Phi_{\tau_i^c} + LET_{\tau_i^c} & \text{if } \tau_i^c \text{ is a LET task.} \end{cases}$$

Proof. An instance of a cause-effect chain $C(j_1, j_2, \ldots, j_n)$ requires by definition that each job in the chain except the header job reads data from the predecessor job. For a job $\tau^c_{k+1}(j_{k+1})$ in the cause-effect chain C to be able to read from the predecessor $\tau^c_k(j_k)$, it is necessary that the read interval of job $\tau^c_{k+1}(j_{k+1})$ overlaps with the data interval of job $\tau^c_k(j_k)$ as was already observed in [2]:

$$RI\left(\tau^c_{k+1}(j_{k+1})\right) \cap DI\left(\tau^c_k(j_k)\right) \neq \emptyset \Rightarrow \tau^c_{k+1}(j_{k+1}) \text{ can read from } \tau^c_k(j_k)$$
$$\text{where } k \in \mathbb{N} : 1 \leq k \leq n-1. \tag{1}$$

Therefore, any selection of jobs $(\tau^c_1(j_1), \tau^c_2(j_2), \ldots, \tau^c_n(j_n))$ with $j_1, j_2, \ldots, j_n \in \mathbb{N}$ which satisfies the data flow constraint in Eq. 1 for all $k \in \mathbb{N} : 1 \leq k \leq n-1$ is potentially an instance of the cause-effect chain C.

It remains to show that the stated interval boundaries are correct.

- The earliest point in time, when a job $\tau^c_i(j_i)$ may read its input data is at its release. By definition, the earliest possible instant of activation is $\underline{\delta}_{\tau^c_i}(j_i)$ (BET task), resp. $(j_i - 1) \cdot P_{\tau^c_i}$ (LET task), to which a constant release offset $\Phi_{\tau^c_i}$ is added.

- If a job $\tau^c_i(j_i)$ of a BET task τ^c_i is timely as required, then the latest possible instant for the job to start execution is by $BCRT_{\tau^c_i}$ time units before the deadline. Thus, this is also the latest possible moment to read the inputs.
  A job $\tau^c_i(j_i)$ of a LET task τ^c_i will always read at the release time.

- Outputs of job $\tau^c_i(j_i)$ can be produced soonest at $BCRT_{\tau^c_i}$ after the earliest point of release given by $\underline{\delta}_{\tau^c_i}(j_i) + \Phi_{\tau^c_i}$ (BET task), resp. at $LET_{\tau^c_i}$ after the release at $(j_i - 1) \cdot P_{\tau^c_i} + \Phi_{\tau^c_i}$ (LET task).

- Outputs of job $\tau^c_i(j_i)$ are available until they are overwritten by the next job $\tau^c_i(j_i + 1)$, which happens latest at $\overline{\delta}_{\tau^c_i}(j_i+1) + \Phi_{\tau^c_i} + WCRT_{\tau^c_i}$ (BET task), resp. $j_i \cdot P_{\tau^c_i} + \Phi_{\tau^c_i} + LET_{\tau^c_i}$.
◀

It is clear that the proposed data flow analysis for a general cause-effect chain C is a template which needs to be enriched which information on the actual cause-effect chain under analysis with regard to the activation model, the relative deadline and the response time bounds. A practical difficulty is that an infinite amount of job combinations must be tested to construct the complete set of possible chain instances $\mathcal{C}$.

Fortunately, the calculation can be simplified for specific settings. Firstly, the special case of a time-triggered cause-effect chain C^{tt} allows to limit the numerical effort to construct due $\mathcal{C}$ to the existence of a hyper period. A related second case is the LET-triggered cause-effect chain C^{let}. Results for C^{tt} and C^{let} are given in Theorem 9. Note that Becker et al. proposed in [1] [2] a similar theorem but here we reduced the set of assumptions with respect to the analyzable system. Becker et al. only allowed single-resource systems with fully periodic task sets with implicit deadlines. Finally, the special case of an event-triggered cause-effect chain C^{et} is discussed, where the termination event of one job activates the successor job in the chain, leads to a restricted set of possible instances of C^{et}.

▶ **Definition 6** (Time-triggered cause-effect chain). *A time-triggered cause-effect chain C^{tt} is a cause-effect chain which has the following properties:*

- *Each task τ^c_i in C^{tt} is a periodically activated BET task with period $P_{\tau^c_i}$ and release offset $\Phi_{\tau^c_i}$.*
- *The deadline of each task τ^c_i in C^{tt} is arbitrary.*
- *At the system start t_0, all tasks in C^{let} are activated.*

▶ **Definition 7** (LET-triggered cause-effect chain). *A LET-triggered cause-effect chain C^{let} is a cause-effect chain which has the following properties:*

- *Each task τ_i^c in C^{let} is a LET task.*
- *The deadline of each task τ_i^c in C^{let} is arbitrary.*
- *At the system start t_0, all tasks in C^{let} are activated.*

With the proposed definition of a LET-triggered cause-effect chain C^{let}, we cover advanced aspects of LET-based system designs

- LET systems with offsets [12]
- LET systems with arbitrary deadlines [4] [12]
- System-level LET systems with multiple resources [6].

▶ **Lemma 8** (Read/data intervals for time/LET-triggered cause-effect chains). *Assume an application $\mathcal{A}$ which is executed on a hardware platform as described in Section 3. Let the application $\mathcal{A}$ consists only of schedulable tasks, i.e., each task satisfies its deadline or LET. Let a path in application $\mathcal{A}$ be a time-triggered cause-effect chain C^{tt}, resp. a LET-triggered cause-effect chain C^{let}, then the boundaries for the read intervals of chain tasks are*

$$\underline{R}\left(\tau_i^c(j)\right) = (j-1) \cdot P_i + \Phi_i$$

$$\overline{R}\left(\tau_i^c(j)\right) = \begin{cases} (j-1) \cdot P_i + \Phi_i + d_{\tau_i^c} - BCRT_{\tau_i^c} & \text{for } C^{tt} \\ (j-1) \cdot P_i + \Phi_i & \text{for } C^{let} \end{cases}$$

and the boundaries for the data intervals of chain tasks are

$$\underline{D}\left(\tau_i^c(j)\right) = \begin{cases} (j-1) \cdot P_i + \Phi_i + BCRT_{\tau_i^c} & \text{if } \tau_i^c \text{ in } C^{tt} \\ (j-1) \cdot P_i + \Phi_i + LET_{\tau_i^c} & \text{if } \tau_i^c \text{ in } C^{let} \end{cases}$$

$$\overline{D}\left(\tau_i^c(j)\right) = \begin{cases} j \cdot P_i + \Phi_i + WCRT_{\tau_i^c} & \text{if } \tau_i^c \text{ in } C^{tt} \\ j \cdot P_i + \Phi_i + LET_{\tau_i^c} & \text{if } \tau_i^c \text{ in } C^{let}. \end{cases}$$

Proof. The results of Theorem 5 for LET tasks can be reused because a LET-triggered chain C^{let} consists exclusively of LET tasks. The interval boundaries for BET tasks in a time-triggered chain C^{tt} can also be derived from Theorem 5, if the constraint of periodic activation instants with release offset is considered. ◀

▶ **Theorem 9** (Data flow analysis for time-triggered and LET-triggered cause-effect chains). *Assume an application $\mathcal{A}$ which is executed on a hardware platform as described in Section 3. Let the application $\mathcal{A}$ consists only of schedulable tasks, i.e., each task satisfies its deadline or LET. Let a path in application $\mathcal{A}$ be the time-triggered cause-effect chain C^{tt}, resp. the LET-triggered cause-effect chain C^{let}.*

The set of potential instances of the time-triggered cause-effect chain C^{tt}, resp. the LET-triggered cause-effect chain C^{let}, with the property that the header job $\tau_1^c(j_1)$ falls into the first hyper period $[0, lcm(P_1, P_2, \ldots, P_n))$ is

$$\mathcal{C}_{HP} = \{C(j_1, j_2, \ldots, j_n) \mid j_1, j_2, \ldots, j_n \in \mathbb{N}$$
$$\wedge (j_1 - 1) \cdot P_1 + \Phi_1 \leq lcm(P_1, P_2, \ldots, P_n)\}$$
$$\wedge \forall k \in \mathbb{N} : 1 \leq k \leq n-1 : RI\left(\tau_{k+1}^c(j_{k+1})\right) \cap DI\left(\tau_k^c(j_k)\right) \neq \emptyset\}$$

Then all instances of a time-triggered cause-effect chain C^{tt}, resp. of a LET-triggered cause-effect chain C^{let}, are contained in the set

$$\mathcal{C} = \{C(j_1 + m \cdot J_1, j_2 + m \cdot J_2, \ldots, j_n + m \cdot J_n) \mid C(j_1, j_2, \ldots, j_n) \in \mathcal{C}_{HP}$$
$$\wedge m \in \mathbb{N} \wedge \forall 1 \leq x \leq n : J_x = \lceil lcm(P_1, P_2, \ldots, P_n)/P_x \rceil\}.$$

Proof. The claim of this proof is that all instances of the time-triggered cause-effect chain C^{tt}, resp. LET-triggered cause-effect chain C^{let} can be constructed from the set $\mathcal{C}_{HP}$. The set $\mathcal{C}_{HP}$ contains all potential instances of C^{tt}, resp. C^{let}, with the property that the header job $\tau_1^c(j_1)$ falls into the first hyper period of the chain tasks $[0, lcm(P_1, P_2, \ldots, P_n))$.

The justification is as follows. It is well known fact that the activation pattern of a periodic task set with/without release offsets repeats after each hyper period. Since the read intervals and data intervals for time/LET-triggered cause-effect chain only depend on activation instants and constants, also the pattern of read intervals and data intervals repeats after each hyper period. This leads to the fact that the pattern of chain instances also repeat after each hyper period.   ◀

▶ **Definition 10** (Event-triggered cause-effect chain). *An event-triggered cause-effect chain C^{et} is cause-effect chain which has the following property:*

- *All tasks in C^{et} are BET tasks without release offset and an arbitrary deadline.*
- *The first task τ_1^c in the chain is activated according to an arbitrary activation model, while any τ_i^c with $1 < i \leq n$ is exclusively activated by the completion event of the predecessor task τ_{i-1}^c.*

▶ **Theorem 11** (Data flow analysis for event-triggered cause-effect chains). *Assume a system as described in Section 3 with a schedulable task set, i.e., each task satisfies its deadline. Then all instances of an event-triggered cause-effect chain C^{et} are contained in the set*

$$\mathcal{C}^{et} = \{C(j_1, j_2, \ldots, j_n) \,|\, j_1, j_2, \ldots, j_n \in \mathbb{N} \,\wedge\, j_1 = j_2 = \ldots = j_n\}.$$

Proof. The condition $j_1 = j_2 = \ldots = j_n$ is a direct consequence of the definition of an event-triggered cause-effect chain.   ◀

5   End-to-End Latency Analysis

Feiertag et al. [7] list different definitions for end-to-end latencies of cause-effect chains. Each definition is relevant in specific contexts, but the most important one is the following:

▶ **Definition 12** (End-to-end latency, also: data age). *The end-to-end latency of an instance of a cause-effect chain, denoted as $Lat(C(j_1, j_2, \ldots, j_n))$, is the maximum amount of time that may elapse from the release of the first job $\tau_1^c(j_1)$ to the completion of the last job $\tau_n^c(j_n)$ in $C(j_1, j_2, \ldots, j_n)$.*

▶ **Definition 13** (Maximum end-to-end latency). *The maximum end-to-end latency $\overline{Lat}(C)$ of a cause-effect chain C is an upper bound on the end-to-end latency of any of its instances*

$$\overline{Lat}(C) = \max_{C(j_1, j_2, \ldots, j_n) \in \mathcal{C}} \{Lat(C(j_1, j_2, \ldots, j_n))\}.$$

The maximum end-to-end latency describes how long it can take until fresh input data at the beginning of the cause-effect chain (e.g. a sensor sample) impacts the output of a cause-effect chain (e.g. actuator control value). This property is important for the reactivity of the system and practical timing requirements often do not relate to the deadline of tasks but in particular to end-to-end deadlines. Tasks deadlines are auxiliary values which emerge in the course of design and implementation.

▶ **Definition 14** (End-to-end deadline). *An end-to-end deadline d_C^{e2e} is the maximum tolerable end-to-end latency of a cause-effect chain C.*

We suppose hereafter that a cause-effect chain C can be decomposed into segments where each segment is either a time-triggered, an LET-triggered or an event-triggered cause-effect chain. This assumption on decomposability is not very restrictive and allows indeed to cover many real-world scenarios.

▶ **Definition 15.** *A cause-effect chain C is called decomposable, if it can be represented by a concatenation of time-triggered, LET-triggered, and event-triggered cause-effect chains C_i, such that $C = (C_1, C_2, \ldots, C_N)$.*

For each type of chain segment, we discuss briefly already known methods from related work how to derive the maximum end-to-end latency.

▶ **Theorem 16** (Maximum end-to-end latency of a time-triggered or LET-triggered chain, [2]). *An upper bound on the end-to-end latency of a time-triggered, resp. LET-triggered, cause-effect chain $C = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$ is*

$$\overline{Lat}(C) = \begin{cases} \max\limits_{C(j_1,\ldots,j_n) \in \mathcal{C}_{HP}} \left\{ \left((j_n - 1) \cdot P_{\tau_n^c} + \Phi_{\tau_n^c} + WCRT_{\tau_n^c}\right) - \left((j_1 - 1) \cdot P_{\tau_1^c} + \Phi_{\tau_1^c}\right) \right\} \\ \qquad \textit{if } C \textit{ is time-triggered} \\ \max\limits_{C(j_1,\ldots,j_n) \in \mathcal{C}_{HP}} \left\{ \left((j_n - 1) \cdot P_{\tau_n^c} + \Phi_{\tau_n^c} + LET_{\tau_n^c}\right) - \left((j_1 - 1) \cdot P_{\tau_1^c} + \Phi_{\tau_1^c}\right) \right\} \\ \qquad \textit{if } C \textit{ is LET-triggered.} \end{cases}$$

Proof. The end-to-end latency of some instance of a time-triggered cause-effect chain, resp. LET-triggered cause-effect chain, is

$$Lat(C(j_1,\ldots,j_n)) = \begin{cases} \left((j_n - 1) \cdot P_{\tau_n^c} + \Phi_{\tau_n^c} + WCRT_{\tau_n^c}\right) - \left((j_1 - 1) \cdot P_{\tau_1^c} + \Phi_{\tau_1^c}\right). \\ \qquad \text{if } C \text{ is time-triggered} \\ \left((j_n - 1) \cdot P_{\tau_n^c} + \Phi_{\tau_n^c} + LET_{\tau_n^c}\right) - \left((j_1 - 1) \cdot P_{\tau_1^c} + \Phi_{\tau_1^c}\right) \\ \qquad \text{if } C \text{ is LET-triggered} \end{cases}$$

which follows directly from Definitions 18, 6, 7. The maximum end-to-end latency is then

$$\overline{Lat}(C) = \max_{C(j_1,\ldots,j_n) \in \mathcal{C}_{HP}} \left\{ Lat(C(j_1,\ldots,j_n)) \right\}.$$

which corresponds to Definition 19 except that the set $\mathcal{C}$ can be reduced to $\mathcal{C}_{HP}$ because we know from Theorem 9 that chain instances repeat after each hyper period. ◀

▶ **Theorem 17** (Maximum end-to-end latency of an event-triggered chain, [11]). *An upper bound on the end-to-end latency of an event-triggered cause-effect chain $C = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$ is*

$$\overline{Lat}(C) = \sum_{k=1}^{n} WCRT_{\tau_k^c}.$$

Proof. By the definition of an event-triggered cause-effect chain, we know that (1) the completion event of one task τ_k^c in the chain C is the activation event of the successor task τ_{k+1}^c, (2) activation and release events coincide. Since the delay between the release of a task until its completion cannot be longer than its worst-case response time, the maximum end-to-end latency of the cause-effect chain C cannot be longer than the sum of the worst-case response times of all tasks in C. ◀

(a) Data flow graph at task level

(b) Data flow graph at job level
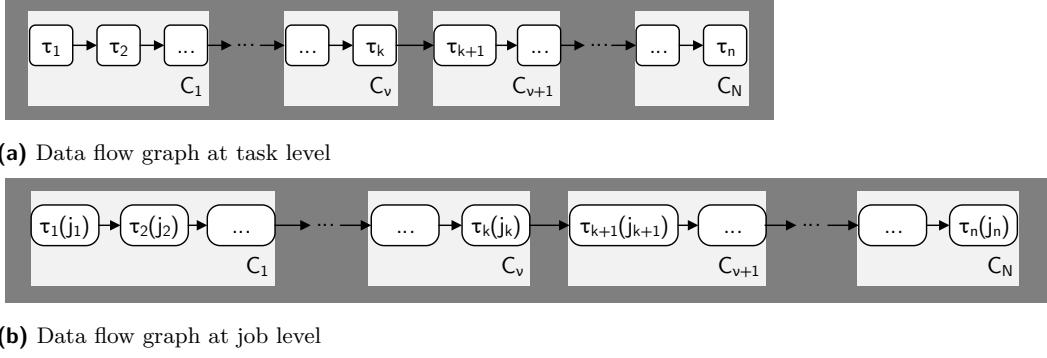
■ **Figure 3** Data flow graph of a decomposable cause-effect chain

A tighter upper bound on the end-to-end latency of an event-triggered chain can be computed by applying methods from, e.g., [17] [10].

To finally derive the maximum end-to-end latency of an entire cause-effect chain, it is a viable approach to compute the maximum end-to-end latencies of the known sub-chains and add the maximum transition latencies between two successive sub-chains.

▶ **Definition 18** (Transition latencies between successive sub-chains). *Let the cause-effect chain C contain two successive sub-chains C_ν and $C_{\nu+1}$. Furthermore, let job $\tau_k^c(j_k)$ be the last job of C_ν and job $\tau_{k+1}^c(j_{k+1})$ be the first job of $C_{\nu+1}$ in an instance of C such that $RI\left(\tau_{k+1}^c(j_{k+1})\right) \cap DI\left(\tau_k^c(j_k)\right) \neq \emptyset$. Then the transition latency between successive sub-chains $TLat(C_\nu(\ldots, j_k), C_{\nu+1}(j_{k+1}, \ldots))$ is the maximum amount of time that may elapse between the completion of job $\tau_k^c(j_k)$ and the release of job $\tau_{k+1}^c(j_{k+1})$.*

▶ **Definition 19** (Maximum transition latency between successive sub-chains). *The maximum transition latency between sub-chains $\overline{TLat}(C_\nu, C_{\nu+1})$ is defined as*

$$\overline{TLat}(C_\nu, C_{\nu+1}) = \max\{TLat(C_\nu(\ldots, j_k), C_{\nu+1}(j_{k+1}, \ldots)) \mid$$
$$j_k, j_{k+1} \in \mathbb{N} \wedge RI\left(\tau_{k+1}^c(j_{k+1})\right) \cap DI\left(\tau_k^c(j_k)\right) \neq \emptyset\}.$$

▶ **Theorem 20** (Maximum end-to-end latency of a cause-effect chain). *Let C be decomposable cause-effect chain, which can can be represented by a concatenation of time-triggered, LET-triggered, and event-triggered cause-effect chains C_i, such that $C = (C_1, C_2, \ldots C_N)$. Then the maximum end-to-end latency of the cause-effect chain C can be bounded from above by*

$$\overline{Lat}(C) \leq \sum_{\nu=1}^{N} \overline{Lat}(C_\nu) + \sum_{\nu=1}^{N-1} \overline{TLat}(C_\nu, C_{\nu+1}).$$

Proof. The end-to-end latency of an instance of a cause-effect chain C can be divided into the end-to-end latencies of its segments $Lat(C_\nu)$ and the transition latency $TLat(C_\nu, C_{\nu+1})$ between the segments. If each of these latency is maximized, we obtain a safe upper of the maximum end-to-end latency of the entire cause-effect chain C. ◀

From Theorems 16 and 17, we already know how to compute the maximum end-to-end latencies of the sub-chains of the decomposable cause-effect chain. In the following theorem, we discuss how to derive the maximum transition latency between successive sub-chains.

C_1 is...	**C_2 is...**		
	C^{tt}	C^{let}	C^{et}
C^{tt}	$TLat(C^{tt}, C^{tt})$	$TLat(C^{tt}, C^{let})$	$TLat(C^{tt}, C^{et})$
C^{let}	$TLat(C^{let}, C^{tt})$	$TLat(C^{let}, C^{let})$	$TLat(C^{let}, C^{et})$
C^{et}	$TLat(C^{et}, C^{tt})$	$TLat(C^{et}, C^{let})$	$TLat(C^{et}, C^{et})$

■ **Table 2** Types of transition latencies between successive sub-chains

▶ **Theorem 21** (Constructing the maximum transition latency between successive sub-chains).
The maximum transition latency between two successive sub-chains $C_\nu, C_{\nu+1}$ in a cause-effect chain C is bounded by

$$\overline{TLat}(C_\nu, C_{\nu+1}) \leq \begin{cases} \overline{\delta}_{\tau_k^c}(2) + WCRT_{\tau_k^c} - BCRT_{\tau_k^c} & \text{if } \tau_k^c \text{ is a BET task} \\ P_{\tau_k^c} & \text{if } \tau_k^c \text{ is a LET task.} \end{cases}$$

where τ_k^c is the last task in C_ν and τ_{k+1}^c is the first task in $C_{\nu+1}$.

Proof. The transition latency between job $\tau_k^c(j_k)$ of sub-chain C_ν and job $\tau_{k+1}^c(j_{k+1})$ of sub-chain $C_{\nu+1}$ starts with the completion of job $\tau_k^c(j_k)$ and ends with the release of job $\tau_{k+1}^c(j_{k+1})$. Since jobs $\tau_k^c(j_k)$, $\tau_{k+1}^c(j_{k+1})$ are successive jobs in an instance of the decomposable cause-effect chain C, we know that $RI\left(\tau_{k+1}^c(j_{k+1})\right) \cap DI\left(\tau_k^c(j_k)\right) \neq \emptyset$. Consequently, the time from the completion of job $\tau_k^c(j_k)$ and the release of job $\tau_{k+1}^c(j_{k+1})$ cannot not be longer than the data interval of job $\tau_k^c(j_k)$, which implies that

$$\forall j_k, j_{k+1} \in \mathbb{N} : RI\left(\tau_{k+1}^c(j_{k+1})\right) \cap DI\left(\tau_k^c(j_k)\right) \neq \emptyset :$$

$$\begin{aligned} TLat(C_\nu, C_{\nu+1}) &\leq \overline{D}\left(\tau_i^c(j)\right) - \underline{D}\left(\tau_i^c(j)\right) \\ &\leq \begin{cases} \overline{\delta}_{\tau_k^c}(2) + WCRT_{\tau_k^c} - BCRT_{\tau_k^c} & \text{if } \tau_k^c \text{ is a BET task} \\ P_{\tau_k^c} & \text{if } \tau_k^c \text{ is a LET task.} \end{cases} \end{aligned}$$

◀

Since a decomposable cause-effect chain C consists only of time-triggered, LET-triggered or event-triggered cause-effect chains, there are nine different types of transition latencies as listed in Table 2. We can profit from this knowledge and refine the results from Theorem 21.

▶ **Corollary 22.**

$$TLat(C^{tt}, C^{tt}) = P_{\tau_k^c} + WCRT_{\tau_k^c} - BCRT_{\tau_k^c}$$

▶ **Corollary 23.**

$$TLat(C^{tt}, C^{let}) = P_{\tau_k^c} + WCRT_{\tau_k^c} - BCRT_{\tau_k^c}$$

▶ **Corollary 24.**

$$TLat(C^{tt}, C^{et}) = \begin{cases} 0 & \text{if Job } \tau_{k+1}^c(j_{k+1}) \text{ is activated by} \\ & \text{the completion event of job } \tau_k^c(j_k) \\ P_{\tau_k^c} + WCRT_{\tau_k^c} - BCRT_{\tau_k^c} & \text{else} \end{cases}$$

▶ **Corollary 25.**

$$TLat(C^{let}, C^{tt}) = P_{\tau_k^c}$$

▶ **Corollary 26.**

$$TLat(C^{let}, C^{let}) = P_{\tau_k^c}$$

▶ **Corollary 27.**

$$TLat(C^{let}, C^{et}) = \begin{cases} 0 & \text{if job } \tau_{k+1}^c(j_{k+1}) \text{ is activated by the completion event of job } \tau_k^c(j_k) \\ P_{\tau_k^c} & \text{else} \end{cases}$$

▶ **Corollary 28.**

$$TLat(C^{et}, C^{tt}) = \overline{\delta}_{\tau_k^c}(2) + WCRT_{\tau_k^c} - BCRT_{\tau_k^c}$$

▶ **Corollary 29.**

$$TLat(C^{et}, C^{let}) = \overline{\delta}_{\tau_k^c}(2) + WCRT_{\tau_k^c} - BCRT_{\tau_k^c}$$

▶ **Corollary 30.**

$$TLat(C^{et}, C^{et}) = \begin{cases} 0 & \text{if job } \tau_{k+1}^c(j_{k+1}) \text{ is activated by} \\ & \text{the completion event of job } \tau_k^c(j_k) \\ \overline{\delta}_{\tau_k^c}(2) + WCRT_{\tau_k^c} - BCRT_{\tau_k^c} & \text{else} \end{cases}$$

6 Robustness of Cause-Effect Chains

A challenge related to software, which includes cause-effect chains, is to design it in a robust manner, so that the required upper bounds on end-to-end latencies are respected even in the presence of software updates. Such software extensions may introduce entirely new tasks or increase the worst-case execution time (WCET) of existing tasks. Thus the amount of workload to be serviced by the system grows and slows down the execution of cause-effect chains which existed prior to the extension. The timing impact of functional extensions can be seen as a disturbance with respect to a cause-effect chain of interest C. A robust software design should be able to tolerate a limited disturbance: meaning that if the disturbance occurs, the cause-effect chain C can still meet its end-to-end deadline.

6.1 Robustness of Time-Triggered Cause-Effect Chains

Consider an application with a time-triggered cause-effect chain $C^{tt} = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$. In the initial software design, each task τ_k^c in C^{tt} has a worst-case response time $WCRT_{\tau_k^c}$ satisfying the task deadline. Moreover, the end-to-end deadline is satisfied in the initial software design. We propose to quantify the impact of a software update as an increase in the worst-case response time of each task τ_k^c in C^{tt} and call it disturbance $\Delta WCRT_{\tau_k^c}$:

$$WCRT'_{\tau_k^c} = WCRT_{\tau_k^c} + \Delta WCRT_{\tau_k^c} \qquad \text{where } \Delta WCRT_{\tau_k^c} \geq 0. \tag{2}$$

To quantify the robustness of the cause-effect chain C^{tt} with regard to disturbances, we introduce the following definitions.

▶ **Definition 31** (Robustness margin for a job in a time-triggered cause-effect chain). *The robustness margin of a job $\tau_k^c(j_k)$ with $k \in \mathbb{N} : 1 \le k \le n-1$ in an instance of a time-triggered cause-effect chain $C(j_1, j_2, \ldots, j_n) = (\tau_1^c(j_1), \tau_2^c(j_2), \ldots, \tau_n^c(j_n))$ is*

$$RM^C(\tau_k^c(j_k)) = \underline{R}\left(\tau_{k+1}^c(q_{\tau_k^c(j_k)} + 1)\right) - \overline{D}\left(\tau_k^c(j_k)\right)$$

where

$$q_{\tau_k^c(j_k)} = \max\left\{x \in \mathbb{N} \mid \underline{R}\left(\tau_{k+1}^c(x)\right) \cap \overline{D}\left(\tau_k^c(j_k)\right)\right\}.$$

▶ **Definition 32** (Robustness margin for a task in a time-triggered cause-effect chain). *The robustness margin of a task τ_k^c in a time-triggered cause-effect chain C is*

$$RM^C(\tau_k^c) = \begin{cases} \min\left\{RM^C(\tau_k^c(j_k)) \mid j_k \in \mathcal{J}_{HP,k}\right\} & \text{if } 1 \le k \le n-1 \\ d_C^{e2e} - \overline{Lat}(C) & \text{if } k = n \end{cases}$$

where

$$\mathcal{J}_{HP,k} = \left\{\tau_k^c(j_k) : j_k \in \mathbb{N} \wedge (\exists C(j_1, j_2, \ldots, j_n) \in \mathcal{C}_{HP} : \tau_k^c(j_k) \in C(j_1, j_2, \ldots, j_n))\right\}.$$

▶ **Theorem 33** (Robustness test). *Let $C = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$ be a time-triggered cause-effect chain with the property that each task τ_k^c in C has a worst-case response time $WCRT_{\tau_k^c}$ satisfying the task deadline $d_{\tau_k^c}$. Moreover, the cause-effect chain C also satisfies its end-to-end deadline d_C^{e2e}. Let $\Delta WCRT_{\tau_k^c} \ge 0$ be the increase in the worst-case response time of each task τ_k^c in C that is caused by a software update.*

The time-triggered cause-effect chain C still satisfies its end-to-end deadline d_C^{e2e} under the increase of the worst-case response times of its tasks τ_k^c by $\Delta WCRT_{\tau_k^c}$, if

$$\forall \tau_k^c \in C : \Delta WCRT_{\tau_k^c} < RM^C(\tau_k^c)$$

The time-triggered cause-effect chain C still satisfies its end-to-end deadline d_C^{e2e} under the increase of the worst-case response times of its tasks τ_k^c by $\Delta WCRT_{\tau_k^c}$ and its task deadlines $d_{\tau_k^c}$, if

$$\forall \tau_k^c \in C : \Delta WCRT_{\tau_k^c} < \min\{RM^C(\tau_k^c), d_{\tau_k^c} - WCRT_{\tau_k^c}\}.$$

Proof. Consider a time-triggered cause-effect chain $C = (\tau_1^c, \tau_2^c, \ldots, \tau_n^c)$ with the property that each task τ_k^c in C has a worst-case response time $WCRT_{\tau_k^c}$ satisfying the task deadline $d_{\tau_k^c}$. Moreover, the cause-effect chain C also satisfies its end-to-end deadline d_C^{e2e}. The data flow analysis proposed in Theorem 9 allows to construct the set $\mathcal{C}_{HP}$, which contains all possible instances of a cause-effect chain C within a hyper period. The pattern of C-instances is repeated after each hyper period, so that it is sufficient to consider $\mathcal{C}_{HP}$ to derive all possible end-to-end latencies and find the maximum. To compute the maximum end-to-end latency of C^{tt}, we have to evaluate all chain instances in the set $\mathcal{C}_{HP}$ according to Theorem 16.

Disturbances $\Delta WCRT_{\tau_k^c}$ can have an impact on the maximum end-to-latency of C according to Theorem 16, if (1) $\Delta WCRT_{\tau_n^c} > 0$ and / or (2) the set $\mathcal{C}_{HP}$ changes. Let us briefly discuss how the set $\mathcal{C}_{HP}$ is modified in the presence of disturbances $\Delta WCRT_{\tau_k^c}$. Increasing the worst-case response of tasks τ_k^c in C by $\Delta WCRT_{\tau_k^c}$

- does not impact the read intervals of tasks τ_k^c,

- does not impact the lower boundary of the data interval of tasks τ_k^c,
- increases the upper boundary of the data interval of tasks τ_k^c.

Consequently, the set of chain instances in one hyper period may comprise more elements under disturbances ($\mathcal{C}'_{HP}$) than in the absence of disturbances ($\mathcal{C}_{HP}$), such that $\mathcal{C}_{HP} \subseteq \mathcal{C}'_{HP}$. Our intention is now to find out how large the disturbances $\Delta WCRT_{\tau_k^c}$ can be such that C^{tt} still satisfies its end-to-end deadline.

From Theorem 16, we can see that

- if $\Delta WCRT_{\tau_n^c} < d_C^{e2e} - \overline{Lat}(C)$, i.e., the disturbance $\Delta WCRT_{\tau_n^c}$ does not become larger than the slack between the end-to-end deadline and the maximum end-to-end latency $\overline{Lat}(C^{tt})$ before the software update, *and*
- if $\Delta WCRT_{\tau_k^c}$ are such that $\mathcal{C}_{HP} = \mathcal{C}'_{HP}$, i.e., no additional chain instances are created,

then C will still satisfies its end-to-end deadline even in the presence of disturbances $\Delta WCRT_{\tau_k^c}$.

It would be useful to have some upper bound $RM^C(\tau_k^c)$ with the property that if $\forall \tau_k^c : 1 \leq k \leq n : \Delta WCRT_{\tau_k^c} < RM^C(\tau_k^c)$, then the two above conditions are fulfilled. We derive the robustness margins $RM^C(\tau_k^c)$ in the following.

With regard to the first condition, we have $\Delta WCRT_{\tau_n^c} < d_C^{e2e} - \overline{Lat}(C)$ which leads to

$$\Delta WCRT_{\tau_n^c} < d_C^{e2e} - \overline{Lat}(C) = RM^C(\tau_n^c).$$

With regard to the second condition, we need to identify under which conditions no new chain instances are created even if the upper boundary of the data interval $\overline{D}\left(\tau_k^c(j_k)\right)$ is increased by $\Delta WCRT_{\tau_k^c}$ for $1 \leq k \leq n-1$. Note already here that the data interval $\overline{D}\left(\tau_n^c(j_n)\right)$ can no have impact because $\tau_n^c(j_n)$ is the last job in the chain.

To begin with, we explore the situation without disturbances.

Let the set of jobs

$$\mathcal{J}_{HP,k} = \{\tau_k^c(j_k) : j_k \in \mathbb{N} \wedge \exists C(j_1, j_2, \dots, j_n) \in \mathcal{C}_{HP} : \tau_k^c(j_k) \in C(j_1, j_2, \dots, j_n)\}$$

corresponds to all jobs of task τ_k^c that are part of the chain instances in $\mathcal{C}_{HP}$.

Let job $\tau_{k+1}^c(q_{\tau_k^c(j_k)})$ be the last job of task τ_{k+1}^c that reads data from job $\tau_k^c(j_k) \in \mathcal{J}_{HP,k}$. As job $\tau_{k+1}^c(q_{\tau_k^c(j_k)} + 1)$ (or any later job of task τ_{k+1}^c) does not read from $\tau_k^c(j_k)$, we have

$$\forall j_k \in \mathcal{J}_{HP,k} : \overline{D}\left(\tau_k^c(j_k)\right) < \underline{R}\left(\tau_{k+1}^c(q_{\tau_k^c(j_k)} + 1)\right)$$
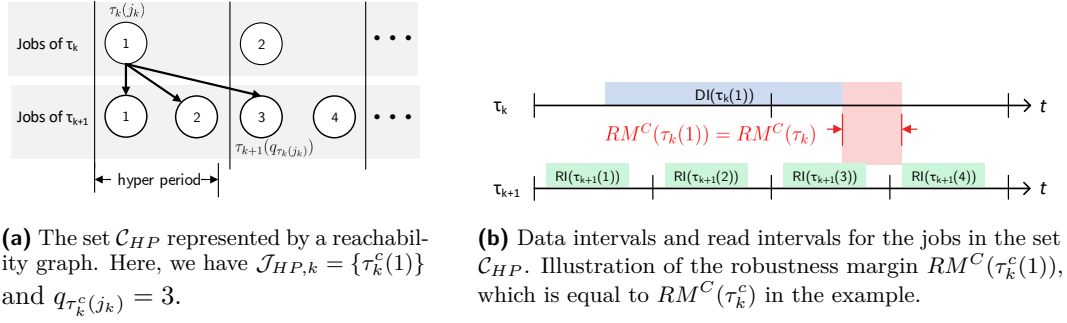
which is equivalent to

$$\forall j_k \in \mathcal{J}_{HP,k} : j_k \cdot P_{\tau_k^c} + \Phi_{\tau_k^c} + WCRT_{\tau_k^c} < ((q_{\tau_k^c(j_k)} + 1) - 1) \cdot P_{\tau_{k+1}^c} + \Phi_{\tau_{k+1}^c}.$$

In the presence of disturbances, the non-overlap of the data interval and read interval is still required if no new chain instances are to be created, such that

$$\forall j_k \in \mathcal{J}_{HP,k} : j_k \cdot P_{\tau_k^c} + \Phi_{\tau_k^c} + (WCRT_{\tau_k^c} + \Delta WCRT_{\tau_k^c}) < ((q_{\tau_k^c(j_k)} + 1) - 1) \cdot P_{\tau_{k+1}^c} + \Phi_{\tau_{k+1}^c}$$

which can be transformed to

$$
\begin{aligned}
\Delta WCRT_{\tau_k^c} &< \min_{j_k \in \mathcal{J}_{HP,k}} \left\{ q_{\tau_k^c(j_k)} \cdot P_{\tau_{k+1}^c} + \Phi_{\tau_{k+1}^c} - j_k \cdot P_{\tau_k^c} - \Phi_{\tau_k^c} - WCRT_{\tau_k^c} \right\} \\
&= \min_{j_k \in \mathcal{J}_{HP,k}} \left\{ \underline{R}\left(\tau_{k+1}^c(q_{\tau_k^c(j_k)} + 1)\right) - \overline{D}\left(\tau_k^c(j_k)\right) \right\} \\
&= \min_{j_k \in \mathcal{J}_{HP,k}} \left\{ RM^C(\tau_k^c(j_k)) \right\} = RM^C(\tau_k^c)
\end{aligned}
$$

(a) The set $\mathcal{C}_{HP}$ represented by a reachability graph. Here, we have $\mathcal{J}_{HP,k} = \{\tau_k^c(1)\}$ and $q_{\tau_k^c(j_k)} = 3$.

(b) Data intervals and read intervals for the jobs in the set $\mathcal{C}_{HP}$. Illustration of the robustness margin $RM^C(\tau_k^c(1))$, which is equal to $RM^C(\tau_k^c)$ in the example.

■ **Figure 4** Example how to compute the robustness margin for a task in a time-triggered cause-effect chain C.

We require therefore that

$$\forall \tau_k^c \in C : \Delta WCRT_{\tau_k^c} < RM^C(\tau_k^c).$$

if C is supposed to satisfy its end-to-end deadline under disturbances $\Delta WCRT_{\tau_k^c}$. ◀

Note that even if the disturbances $\Delta WCRT_{\tau_k^c}$ that will be induced by a planned software update are not known exactly beforehand, the robustness margins allow to predict which tasks are critical in preserving the timeliness of the cause-effect chain.

6.2 Dependent Time-Triggered Cause-Effect Chains

Assume there are several cause-effect chains in an application which have dependencies due to common tasks (cf. Figure 2a where cause-effect chains C, D contain both task τ_3), then a robustness margin for a common task τ_i can be defined as follows.

▶ **Definition 34.** *The robustness margin of a task τ_i in an application with only time-triggered cause-effect chains is*

$$RM(\tau_i) = \min\{RM^C(\tau_k^c) : \tau_k^c = \tau_i \wedge \tau_k^c \in C \in \mathcal{A}\}.$$

Clearly, the end-to-end latencies of all time-triggered cause-effect chains in an application are respected under disturbances $\Delta WCRT_{\tau_k^c}$ if

$$\forall C : \forall \tau_k^c : 1 \leq k \leq n : \ \Delta WCRT_{\tau_k^c} < RM(\tau_k^c).$$

6.3 Robustness of LET-Triggered Cause-Effect Chains

LET-triggered cause-effect chains have an inherent robustness because any change in the actual response time of a task τ_k^c is hidden by the logical execution time $LET_{\tau_k^c}$ under the condition that the response time does not exceed the logical execution time. If, however, this condition is violated, then it is an option to increase the logical execution time. Thus the problem is whether it is possible to increase the logical execution times $LET_{\tau_k^c}$ of a given software design by $\Delta LET_{\tau_k^c}$

$$LET'_{\tau_k^c} = LET_{\tau_k^c} + \Delta LET_{\tau_k^c} \qquad\qquad \text{where } \Delta LET_{\tau_k^c} \geq 0. \tag{3}$$

while still satisfying the end-to-end deadline. It possible to re-use the above presented robustness analysis, but a small modification must be made: In contrast to the case of time-triggered cause-effect chain, adding $\Delta LET_{\tau_k^c}$ to $LET_{\tau_k^c}$ increases only the upper boundary
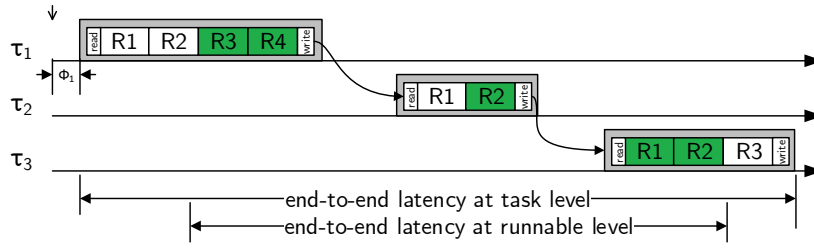
Figure 5 Cause-effect chain at runnable level

but also the lower boundary of the data interval of task τ_k^c. A possible solution is to set $\underline{D}\left(\tau_k^c(j_k)\right) := (j_k - 1) \cdot P_{\tau_k^c} + \Phi_i$ which is also a safe lower but less tight boundary of the data interval of job $\tau_i^c(j)$. This will lead to more paths in the data flow analysis (larger super set of possible paths), but this does not impact the accuracy of the computed maximum end-to-end latency according to Lemma 3 in [2].

6.4 More Detailed Abstraction Level

In automotive software architecture, a hierarchical task model is common where a task is composed of runnables. If the abstraction level of runnables is applied in the system model, an instance of a cause-effect chain is not a sequence of jobs but a sequence of runnable-instances as shown in Figure 5. A possible definition of an end-to-end latency at the runnable level is as follows.

▶ **Definition 35** (End-to-end latency at runnable level). *The end-to-end latency of an instance of a cause-effect chain modeled at runnable level is the maximum amount of time that may elapse from the release of the first runnable to the completion of the last runnable in the chain instance.*

With regard to the end-to-end latency, the precision gained at the runnable level compared to the task level is not considerable and decreases with the number of tasks in the cause-effect chain. The difference in precision is even smaller, if another definition of an end-to-end latency at the runnable level is chosen which includes, for instance, the reading phase of the first task and/or the writing phase last task in the chain.

7   Experiments

7.1   Case Study

Collaboration with Max-Jonas Friese. To be continued.

ECM   PtGw   TCM

ECM CAN   TCM CAN
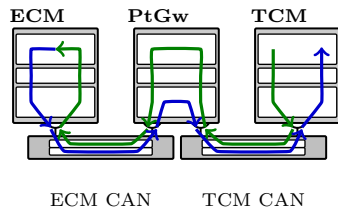
To test our approach we used a case-study with a set-up commonly found in automotive powertrains. It features three ECUs connected via two CAN-FD networks as depicted in Figure **??**. Furthermore, it contains realistic data for one cause-effect chain spanning over three ECUs. The CPC is the central powertrain controller which acts as a proxy for the powertrain domain but also manages the inner-domain communication between the transmission control module (TCM) and the engine control module (ECM). The case-study comprises data for communication artifacts on the CANs, as well as for the software deployed on the ECUs. In this, it contains a challenge commonly found in industry: incomplete data. During the different stages of system engineering implementation details are often not available, e.g. because a supplier is responsible for parts of an ECUs software. For the analysis of end-to-end latencies, the temporal behavior of these parts must be approximated with the help of suitable models.

The cause-effect chain we analyze is part of the torque request and deliver functionality of the powertrain. More precisely, it the chain which is set off when the transmission controller needs to reduce torque, e.g. at a gear change. For this, a signal is generated in the TCM and send in a frame on the TCM CAN to the CPC. The CPC checks the requests before forwarding it the ECM. The ECM then tries to reduce torque to the extend possible and answers to the CPC with the torque reduction actually provided. The answer of the ECM is directly routed from the ECM CAN to the TCM CAN for fast reception at the TCM. The cuase-effect chain ends with the occurrence of the updated value for the provided torque in the memory of the TCM.

8   Conclusion

Acknowledgment

References

1   Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas 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)*, pages 159–169. IEEE, 2016.

2   Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104–113, 2017.

3   Matthias Becker and Saad Mubeen. Timing analysis driven design-space exploration of cause-effect chains in automotive systems. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 4090–4095. IEEE, 2018.

4   Christian Bradatsch, Florian Kluge, and Theo Ungerer. Data age diminution in the logical execution time model. In *International Conference on Architecture of Computing Systems*, pages 173–184. Springer, 2016.

5   Marco Dürr, Georg Von Der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. End-to-end timing analysis of sporadic cause-effect chains in distributed systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):58, 2019.

6   Rolf Ernst, Leonie Ahrendts, and Kai-Björn Gemlau. System level let: Mastering cause-effect chains in distributed systems. In *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*, pages 4084–4089. IEEE, 2018.

7   Nico Feiertag, Kai Richter, Johan Nordlander, and Jan 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.

8   Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, pages 301–310. IEEE, 2010.

9   Julien Forget, Frédéric Boniol, David Lesens, and Claire Pagetti. A real-time architecture design language for multi-rate embedded control systems. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 527–534. ACM, 2010.

10   Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia, and Nicolas Sordon. Improving and estimating the precision of bounds on the worst-case latency of task chains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2578–2589, 2018.

11   Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis–the symta/s approach. *IEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

12   Julien Hennig, Hermann von Hasseln, Hassan Mohammad, Stefan Resmerita, Stefan Lukesch, and Andreas Naderlinger. Towards parallelizing legacy embedded control software using the let programming paradigm. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–1. IEEE, 2016.

13   Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer Science & Business Media, 2001.

14   Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. Analytical characterization of end-to-end communication delays with logical execution time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2244–2254, 2018.

15   Claire Pagetti, Julien Forget, Frédéric Boniol, Mikel Cordovilla, and David Lesens. Multi-task implementation of multi-periodic synchronous programs. *Discrete event dynamic systems*, 21(3):307–338, 2011.

16   Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, 2015.

17   Johannes Schlatow and Rolf Ernst. Response-time analysis for task chains in communicating threads. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–10. IEEE, 2016.

18   Johannes Schlatow, Mischa Mostl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf 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)*, pages 1–9. IEEE, 2018.