Estimating End-to-End Latencies in Automotive Cyber-physical Systems

Max J. Friese
Mercedes-Benz AG
Sindelfingen, Germany
Department of Computer Science
Kiel University
Kiel, Germany
max_jonas.friese@daimler.com

Dirk Nowotka
Department of Computer Science
Kiel University
Kiel, Germany
dn@informatik.uni-kiel.de

Controller networks in today's automotive systems consist of more than 100 ECUs connected by various bus protocols. Seamless operation of the entire system requires a well-orchestrated interaction of these ECUs. Consequently, to ensure safety and comfort, a performance analysis is an inherent part of the engineering process. Conducting such an analysis manually is expensive, slow, and error prone. Tool support is therefore crucial, and a number of approaches have been presented. However, most work is limited to either network latencies or software latencies which results in an analysis gap at the transition between different layers of the communication stack. The work presented here introduces an approach to close this gap. Furthermore, we discuss the integration of different methods to obtain an end-to-end latency analysis.

1 Introduction

Distributed cyber-physical systems (CPS) are the key for many technology developments in today's connected world, one prominent example is the development of automated vehicles. Advancing *classic* embedded computing, CPS stand for the integration of computing and physical processes in networks of heterogeneous components. Due to their distributed nature, system functions span over multiple devices and possibly contain feedback loops [22]. Two aspects need to be considered to assess the correct behavior of a CPS with regards to a system function: *how* does the system react and *when* does the system react. More precisely, if a certain stimulus occurs, the system has to react correctly and at the correct time, e.g. an automated car should start to brake immediately, if its sensors detect a pedestrian on the road. Consequently, the time to react is part of the requirements for the correct behavior.

Accordingly, the specification and validation of the temporal behavior of CPS is an important part of system's engineering. The latter is getting considerably more complex due to the continuing growth of hard- and software architectures. The former usually starts with an end-to-end time budget for a system function, e.g. the backup camera should show an image within 300 ms after the car was put into reverse. To verify that the system meets the timing constraints for a function, one or more cause-effect chains are analyzed. Each cause-effect chain describes one flow of data through the system in detail. The cause-effect chains are further subdivided and broken down into smaller parts, e.g. the sub-chain within one electronic control unit (ECU). Each sub-chain contributes to the end-to-end latency and receives a share of the timing budget accordingly. The implementer of the different sub-chains have to assure that their part of the chain meets its timing constraints.

The latency caused within the *cyber* part of the system falls into one of the following categories: (1) latencies due to processing of software, or (2) latencies due to network communication. For the software

A. Fehnker and H. Garavel (Eds.):
Models for Formal Analysis of Real Systems (MARS 2020)
EPTCS 316, 2020, pp. 134–148, doi:10.4204/EPTCS.316.6

part, the possible core execution times of the involved tasks and the dataflow through the tasks is analyzed to find the path which yields the worst-case latency. The latency induced by network communication is the sum of the so-called transmission and the so-called propagation delay. In summary: the analysis of software latencies ends at the last write access on some variable in global memory and the analysis of network latencies starts at the transmission of a network frame.

In this work we argue that meeting local timing constraints is no sufficiency for meeting end-to-end timing constraints if partitioning of the chain is not done carefully. In this context we call attention to an important aspect currently not sufficiently considered in the integration of software- and network analyses. It originates in the need for increased bandwidth which lead to a relaxation in the static mapping of protocol data units (PDUs) into their encapsulating PDUs. To increase flexibility and thereby save network resources, e.g. by sending two encapsulated PDUs alternately, PDU mapping is done dynamically. The actual mapping is determined at runtime. On the data link layer, PDUs are triggered for sending either when full, when certain timers expire, or when a higher-layer PDU needs to be sent immediately. The combination of event-based packing, immediate sending, and dynamic mapping leads to complex situations where it is not directly evident whether an updated value is sent with the very next encapsulating PDU. The impacts of these mechanisms on end-to-end timings are currently neither considered in software-focused nor in network-focused analyses, leaving a gap in the methodology for formally derived, safe end-to-end estimations.

1.1 Contribution

The contributions of this paper are twofold. First, we present an analysis model and technique for the temporal behavior of PDU-transmission mechanisms, which are commonly used in automotive CPS. Secondly, we discuss the integration of formal methods to obtain end-to-end estimations. In particular we

- introduce *timing models* and show how two or more can be combined to model the composite of temporal behaviors.
- present an approach to encode different trigger mechanisms for signals, PDUs and frames to obtain safe estimations on their impact on end-to-end latencies.
- describe how to integrate this approach with analyses for software-level estimations to obtain end-to-end latencies.
- report the applicability on an industrial-scale use case.

1.2 Outline

This paper is organized as follows: related work is presented in the next section. Subsequently, we describe the systems considered here in Section 3. In Section 4.2 we present our approach to estimate the impact of different PDU triggering mechanisms on end-to-end timing. We use CAN-FD as an example for the data link layer and show how to plug-in respective analyses. In Section 4.4 we discuss how to integrate the approach of the previous section with existing approaches for the software part to obtain an end-to-end latency. To demonstrate applicability, we report the application of the approach on an automotive use case in Section 5. Finally, in Section 6 and Section 7 we provide an insight into planned future work and conclude.

2 Related Work

Related work comes from two categories. Firstly, these are approaches to estimate the latency due to network communication, i.e. the delay between a network controller of one ECU and the network controller of the next ECU in the chain. Respective analyses have been presented for different automotive field busses [8, 24]. So-called *holistic* approaches are additionally concerned with parts of the ECU's software [19]. However, currently they rely on single-core analyses [20]. Earlier, so-called *compositional* approaches were developed to cope complexity and therefore make analysis applicable. In compositional performance analysis (CPA) different local scheduling analyses are combined to obtain end-to-end estimations [25]. CPA got a lot of research attention, and was used e.g. for the end-to-end response time analysis in automotive systems [27]. However, in its basic form, CPA suffers from the problem that multiple worst-cases are possibly considered simultaneously although they can not occur at the same time. Recent work in the area underlines applicability for industrial-scale use cases [13, 28] and improves, i.e. reduces, pessimism of the estimations [18].

Related work from the second category is concerned with estimations for the time needed to propagate signal changes through the software within an ECUs. For automotive CPS, this analysis is performed by analyzing the end-to-end response time of task chains. Two types of task chains are distinguished. On the one hand, there are chains where tasks activate their successors based on events generated by changes in signal values [26]. These chains are also referred to as *functional chains* [14]. On the other hand, there are chains through tasks activated by periodic timers; these are so-called *cause-effect chains*. For the sake of simplicity, we will refer to both kinds of chains as cause-effect chains in this work. Two latency semantics are of interest for cause-effect chains: *response time* is the time needed to react to a certain input, and *data age* is the time span an input has still an impact on the output [11]. Like functional chains, cause-effect chains in software are well-studied in terms of algorithmic approaches to estimate latencies [3, 4]. Other approaches to tackle this problem include encoding of the system's behavior in integer linear programming [5, 21] and constraint programming [12]. Furthermore, the work around the logical execution time paradigm (LET) [16] must be attributed, as it aims to simplify timing analysis through deterministic behavior, also on system-level [10]. Corresponding work was done in researching LETs potential to determine end-to-end latencies [15, 23].

However, integrating models from different development stages, and therefore with different levels of details to determine end-to-end latencies in all stages of systems engineering is still an ongoing challenge. The constraint programming (CP) approach followed in this paper is especially promising because the level of detail can easily adjusted by adding or removing constraints.

3 System Model

Network clusters in modern automotive systems are usually divided into multiple domains focused on a set of functionalities of the car, e.g. powertrain or infotainment [17]. Each domain comprises a set of controllers collaborating to implement system functions. Additionally, gateway controllers encapsulate the communication with other domains. The topology of the network is also reflected in the communication design. Communication can be divided into two categories. On the one hand there are classic field busses like LIN, CAN or FlexRay which are compatible with static resource allocation of real-time embedded systems. In current architectures they are commonly used to interconnect ECUs within a domain. Due the increasing amount of bandwidth-intense applications, like e.g. image recognition, domain controllers, on the other hand, are connected via Ethernet backbones. The Ethernet deployment coincides

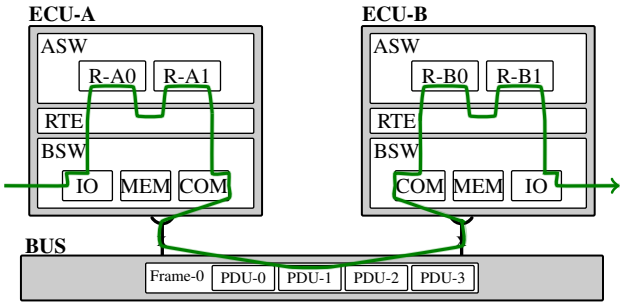

Figure 1: Cause-effect Chain (green) spanning over two ECUs

with a paradigm shift from signal-based to service-oriented communication, especially for inter-domain communication. In signal-based communication the possible data exchange between ECUs is static and concluded after design, whereas service-oriented communication allows to extend communication dynamically. However, both paradigms have in common that PDUs are used for the vertical communication through the stack within an ECU. Different mechanisms exists to trigger the transmission of PDUs. After a PDU was triggered, e.g. by a timeout, it is encapsulated in a lower-layer PDU. Once the data link layer is reached, PDUs are put on the physical medium by the communication controller of the ECU. Nowadays, for some PDUs, encapsulation is decided dynamically at runtime. As a consequence, an update in a signal value is not necessarily sent in the next available PDU if another PDU is first in line. The postponed PDU is sent with a subsequent transmission. This leads to complex scenarios aggravating a manual verification of the end-to-end latency constraints of distributed system functions.

Within an ECUs the data flow for computing a system function passes multiple so-called runnables as depicted in Figure 1 (e.g. R-A0). A runnable is a piece of software dedicated to the computation to parts of the overall functionality. For processing, multiple runnables are combined to a task. Task sets of automotive ECUs comprise multiple, periodically activated tasks with different rates and offsets. Additionally, sporadically activated task handle interrupts, e.g. for sensor readings. The signals used to exchange values are stored in variables located in a global memory.

At the bound of the ECU and the network, dedicated communication tasks copy data from global memory to the buffer of the communication controller if transmission requested. In this work, we divide the cause-effect chains associated to a time-constrained system function into two kinds of sub-chains: (1) communication task to communication task via network and (2) communication task to communication task via task chain. The course of events for both kind of chain is mainly driven the possible time intervals memory is accessed by the runnables of a task. In this work, we consider implicit data reception and transmission on task-level [2], meaning that a task receives all signal values when it is activated and that all signal values are written collectively when the task terminates.

We describe how to obtain estimations for the first kind of sub-chain in Section 4.2. The problem of estimating latencies for task chains is well researched. We briefly revisit the problem in Section 4.3 in order to discuss how to obtain *overall* end-to-end latencies in Section 4.4.

4 Formal abstractions

The temporal behavior of the system with regard to a cause-effect chain depends on the possible time intervals for different events, e.g. a PDU being triggered for sending. The trigger mechanism for PDUs

are described in detail in the next section. However, less specifically speaking, a PDU can be triggered by a timer or due to a value change in a signal. As described above, the latter depends on the points of time a task produces an update for the signal value. Although signals are likely to be produced by a dedicated runnable and therefore written by a single task, in general it is possible that multiple tasks write a signal. To describe and combine intervals of time in which this might happen, we introduce the notion of *Timing Models*. Our formalization is based on a discrete and finite time model: let $\mathbb{T} = \{ T_{\min}, \ldots, T_{\max} \} \subseteq \mathbb{N}$ be the time domain with $T_{\min}$ and $T_{\max}$ as minimum and maximum valid points of time. Furthermore, let $T_{\sup} = \sup(\mathbb{T})$ be a value to indicate invalid points of time.

4.1 Timing Models

To model the temporal behavior of the system that is being analyzed, we consider different events, like e.g. a PDU being triggered for sending. Similar to the *Arrival Curves* introduced in [7] we use *Timing Models* to describe the nature of how events occur. However, unlike the *Cumulative Function* of *Network Calculus* [6] or the *Interval Bound Functions* of *Real-time Calculus* [29] we do not use them to derive request and response counts but use them to bound the interval of time in which an event might occur within $\mathbb{T}$.

Definition 1 (Timing Model)**.** *A timing model is a function $m : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ which maps the occurences of an event to the first and the last possible point of time the event might occur. For elements of the codomain we use the projection functions π_1 and π_2 to access the respective element, e.g. let $p = (a,b) \in \mathbb{N} \times \mathbb{N}$, then $\pi_1(p) = a$ and $\pi_2(p) = b$.*

We distinguish two types of timing models. Firstly, *Periodic Timing Models* are used to describe events triggered by periodic clocks. The time in which these events might occur does not vary. Secondly, *Sporadic Timing Models* allow for the specification of temporally less predetermined events. Here, the time span for an occurrence of the event can only be be narrowed down to a minimum and maximum interarrival time.

Definition 2 (Periodic Timing Models)**.** *A periodic timing model $t^{\mathbf{P}}_{o,p,n} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is a timing model parameterized with three arguments $o \in \mathbb{N}$ and $p,n \in \mathbb{N}_{>0}$ with $t^{\mathbf{P}}_{o,p,n}(i) = (a_i, a_i)$ where $a_i = o + \lceil \frac{i}{n} \rceil \cdot p$ for all $i \in \mathbb{N}$. The family of periodic timing models is defined as $\mathbf{TMP} = \{ t_{o,p,n} \mid o \in \mathbb{N}, p,n \in \mathbb{N}_{>0} \}$.*

Definition 3 (Sporadic Timing Models)**.** *A sporadic timing model $t^{\mathbf{S}}_{l,u,n} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is a timing model parameterized with three arguments $l,n \in \mathbb{N}_{>0}$ and $u \in \mathbb{N}_{>l}$ with $t^{\mathbf{S}}_{l,u,o}(i) = (l_i, u_i)$ where*

$$l_i = \lceil \tfrac{i}{n} \rceil \cdot l \;\; \textit{and} \;\; u_i = \left(\lceil \tfrac{i}{n} \rceil + 1 \right) \cdot u$$

for all $i \in \mathbb{N}$. The family sporadic timing models is defined as $\mathbf{TMS} = \{ t^{\mathbf{S}}_{l,u,n} \mid l,n \in \mathbb{N}_{>0}, u \in \mathbb{N}_{>l} \}$

Let $\mathbf{TM} = \mathbf{TMP} \cup \mathbf{TMS}$ be the set of timing models. Finally, to combine multiple possible time intervals for an event, we define the $\sqcup$-operator for timing models.

Definition 4 (Union of Timing Models)**.** *Let $t_0, t_1 \in \mathbf{TM}$ be timing models. To define the union of t_0 and t_1, formally $t_0 \sqcup t_1$, we distinguish three cases:*

Case 1 *$t_0 = t^{\mathbf{P}}_{o_0,p_0,n_0}$ is a periodic timing model and $t_1 = t^{\mathbf{P}}_{o_1,p_1,n_1}$ is a periodic timing model.*

$$t_{o_0,p_0} \sqcup t_{o_1,p_1} = \begin{cases} t^{\mathbf{P}}_{o,p',n'} & \textit{if } o_0 = o_1 \wedge \mathrm{mod}(p_{max}, p_{min}) = 0 \\ t^{\mathbf{S}}_{l,u,n'} & \textit{else} \end{cases}$$

with $p_{min} = \min\{ p_0, p_1 \}$, $p_{max} = \max\{ p_0, p_1 \}$, $l = \min\{ p_0, p_1 \}$, $u = \max\{ o_0, o_1 \} + p_{max}$, $n' = n_0 + n_1$, and $p' = p_{min}$.

Case 2 *$t_0 = t^{\mathbf{S}}_{l,u,n_0}$ is a sporadic timing model and $t_1 = t^{\mathbf{P}}_{o,p,n_1}$ is a periodic timing model.*

$$t^{\mathbf{S}}_{l,u,n_0} \sqcup t^{\mathbf{P}}_{o,p,n_1} = t^{\mathbf{P}}_{o,p,n_1} \sqcup t^{\mathbf{S}}_{l,u,n_0} = t^{\mathbf{S}}_{l',u',n'}$$

with $l' = \min\{l, p\}$, $u' = \max\{o+p, u\}$, and $n' = n_0 + n_1$.

Case 3 *$t_0 = t^{\mathbf{S}}_{l_0,u_0,n_0}$ is a sporadic timing model and $t_1 = t^{\mathbf{S}}_{l_1,u_1,n_1}$ is a sporadic timing model.*

$$t^{\mathbf{S}}_{l_0,u_0,n_0} \sqcup t^{\mathbf{S}}_{l_1,u_1,n_1} = t^{\mathbf{S}}_{l',u',n'}$$

with $l' = \min(\{l_0, l_1, |l_0 - l_1|\})$, $u' = \max(\{u_0, u_1\})$, and $n' = n_0 + n_1$.

The $\sqcup$-operator allows us to combine arbitrary timing models for events. This is particularly suitable for describing all possible points of time at which a signal value might be updated. Consider a signal written by two runnables processed in two different tasks. One task is activated every 5 ms and without an offset and the other one is activated every 10 ms with an offset of 2.5 ms. This means, updates of the signal will happen after the following points of time:
first:0 ms second:2.5 ms third:5 ms fourth:10 ms fifth:12.5 ms sixth:15 ms.

This pattern is safely approximated by the sporadic activation pattern $t^{\mathbf{S}}_{5000,12500,2}$ which gives the following possible intervals for the i^{th} update:

$$t^{\mathbf{S}}_{5000,12500,2}(1) = (0, 12500) \quad t^{\mathbf{S}}_{5000,12500,2}(2) = (0, 12500) \quad t^{\mathbf{S}}_{5000,12500,2}(3) = (5000, 25000)$$
$$t^{\mathbf{S}}_{5000,12500,2}(4) = (5000, 25000) \quad t^{\mathbf{S}}_{5000,12500,2}(5) = (10000, 37500) \quad t^{\mathbf{S}}_{5000,12500,2}(6) = (10000, 37500)$$

Safely here means, that the possible point of time for the n^{th} activation lies within the value of the timing model for n. In the next section we use this to describe the complex temporal behavior of different PDU triggers based on the potential updates of signal values.

4.2 Estimations on Network-level

Based on the system model described in Section 3 our formal model comprises four types of elements: (1) signals, (2) PDUs, (3) frames and (4) communication tasks. For each element type we distinguish two types of variables: (1) input parameter, and (2) modeling variables. The input parameter of a task include a timing model for its activation and a deadline. The input parameter for a signal consists of a single timing model and a reference to a PDU. Signal changes are generated by the runnables of the transmitting ECU. For each producing runnable, a timing model describing the points of time the signal value possibly changes is derived from the activation model of the task. These timing models are *summed up* with the help of the $\sqcup$-operation. The resulting timing model is the aforementioned input. The input parameters of a PDU are more diverse. First of all, the different triggering options have to be considered. Besides a direct triggering by contained signals, this can be a threshold for the filling level, and a timeout. Furthermore, for PDUs which are encapsulated in dynamically filled container PDUs, the *collection semantics* are needed to describe possible behaviors. The collection semantics can either be *last-is-best* or *queued*. Queued collection semantics guarantees that every instance of the contained PDU is visible on the wire (cf. [1]). Thirdly, the maximum length and the size of the threshold for triggering need to be known to determine a triggering due to the filling level. Finally, a reference to the encapsulating frame is included in the set of input parameters for a PDU. The input parameters of a frame comprise its priority for arbitration, its length and a reference to the communication task responsible for its transmission. These parameters are specific for CAN FD and might need to be adjusted for other physical layer protocols.

Table 1: Time Variables of Network Model Elements

Element	Variable	Event description
Signal	$\varphi^S_{i,j}$	The time span for the j^{th} change of signal i
PDU	$\alpha^P_{i,j}$	The time span in which the j^{th} instance of PDU i is triggered
PDU	$\sigma^P_{i,j}$	The time span in which the j^{th} instance of PDU i is moved to the lower level buffer
Frame	$\alpha^F_{i,j}$	The time span in which the j^{th} instance of frame i is triggered
Frame	$\sigma^F_{i,j}$	The time span in which the j^{th} instance of frame i is tried to be sent by its sending communication task
Frame	$\varepsilon^F_{i,j}$	The time span in which the j^{th} instance of frame i is fully received by its receiving communication task
Task	$\alpha^T_{i,j}$	The time span for the activation of the j^{th} instance of task i
Task	$\varepsilon^T_{i,j}$	The time span for the completion of the j^{th} instance of task i

Besides its input parameters, each model element has different types of modeling variables subjected to the modeling constraints. Firstly, there a the *time* related variables listed in Table 1. Secondly, each instance of any element is encapsulated in an instance of a lower-layer element. This is modeled in the parameter n for the types signal (n^S), PDU (n^P) and frame (n^F). It contains a reference to the instance of the container for each occurrence of the respective element. For frames the semantic is slightly different as n contains the instance of the transmitting communication task in this case. Thirdly, in order to obtain safe estimations, a minimum amount of occurrences of each element needs to be considered. Therefore, assume that $\mathbb{T}$ covers a sufficient period of time in which all combinations of relative offsets between occurrences of the timing models appear. Bounds on the length of this period are discussed below. The maximum number of occurrences can be computed for most model elements, if this time interval is fixed. For all the remaining elements, i.e. the container PDUs, the number of occurrences has to be derived from the occurrences of the contained elements. Let $\Omega^S_i = \{\,\text{occ\_min}^S_i, \ldots, \text{occ\_max}^S_i\,\}$ be the index set for the occurrences of signal i. Let Ω^P, Ω^F, and Ω^T hold the index set of occurrences for PDUs, frames, and tasks respectively.

The update of a signal value is modeled with the help of a timing model as described above. Let t^S_i be the timing model of signal i. Note that t^S_i can be sporadic or periodic. The S indicates that it belongs to a signal here. The following constraint is added to the model for all $j \in \Omega^S_i$:

$$\varphi^S_{i,j} \geq \pi_1\left(t^S_i(j)\right) \wedge \varphi^S_{i,j} \leq \pi_2\left(t^S_i(j)\right) \tag{1}$$

When a signal value was updated, the so-called *update bit* is set. The update of the value then eventually triggers the sending of a PDU. However, since we are interested in the transmission time for the changed value, we also need to model in which occurrence of its designated container (PDU^S) the update is transmitted. This is done by adding the following constraints for all signals i and their occurrences $j \in \Omega^S_i$:

$$\forall k \in \Omega^P_\ell : (\varphi^S_{i,j} > \sigma^P_{\ell,k-1} \wedge \varphi^S_{i,j} \leq \sigma^P_{\ell,k}) \rightarrow (n^S_{i,j} = k)\,. \tag{2}$$

As described above two different events have to be considered for the triggering of PDUs : triggering due to timeout modeled by α^{P-T}, and triggering due to transmission request by containees modeled by α^{P-E}. If no clock triggering is configured, α^{P-T} is set to T_{sup}. Analogously, if a PDU is not triggered by any containee, α^{P-E} is set to T_{sup}. The containees of a non-container PDU i are signals. Accordingly, the following constraint is added for all $j \in \Omega^P_i$:

$$\alpha^{P-E}_{i,j} = \min\{\,\varphi^S_{\ell,k} \mid \ell \in \text{sigs}^P_i, k \in \Omega^S_\ell \wedge n^S_{\ell,k} = j\,\}\,. \tag{3}$$

For container PDUs, possibilities for triggering are more complex. The following points of time are considered conditionally: the point of time the first containee was triggered $\alpha_{i,j}^{P-C1}$, the point of time the first containee was triggered plus the timeout of the container $\alpha_{i,j}^{P-CT}$, and the point of time the length of the contained PDUs exceeds the threshold of the container $\alpha_{i,j}^{P-Cn}$. To detect a triggering of a container PDU due to exceeding of the threshold, the filling level needs to be determined. To this end, we introduce an additional variable foreach instance j of a PDU i, $len_{i,j}^{P}$, which contains the total length of the PDU. Additionally, for a pair of PDUs i and j and each instance j of i and k of ℓ we add an auxiliary variable $c_{i,j,\ell,k}^{P}$ which holds 1 if k is contained in j and 0 otherwise, i.e.

$$c_{i,j,\ell,k}^{P} = \begin{cases} 1 & \text{if } \ell \in \text{PDU}_i^P \wedge n_{\ell,k}^P = j \\ 0 & \text{else} \end{cases} \tag{4}$$

for all PDUs i, ℓ and $j \in \Omega_i^P$, $k \in \Omega_i^L$.

Depending on the collection semantics, an instance of a PDU might be overwritten if its container is not send between two updates. This means, that c^P is 1 for two instances of the contained PDU. To model the fact that an instance k of a PDU ℓ might be overwritten by a subsequent instance in the instance j of its container PDU i, we add an additional variable $o_{i,j,\ell,k}^{P}$ with

$$o_{i,j,\ell,k}^{P} = \begin{cases} 1 & \text{if } c_{i,j,\ell,k}^{P} = 1 \wedge \exists k' \in \Omega_\ell^P \colon k' > k \wedge n_{\ell,k'}^P = j \\ 0 & \text{else} \end{cases} \tag{5}$$

for all PDUs i, ℓ and $j \in \Omega_i^P$, $k \in \Omega_\ell^P$.

The length of a non-container PDU is fixed, based on the contained signals and a fixed-size header. The length of a container PDU depends on its PDU layout. If it has a *static* layout, the length is fixed. Otherwise, if it has a *dynamic* layout, the collection semantics of the contained PDUs is the crucial factor. If the collection semantic is *last-is-best* the content in the container can be overwritten. Otherwise, if the collection semantic is *queued*, multiple instances of the same PDU can be transmitted in one container. Note, that we assume that containers cannot be nested. The actual length of a container can therefore finally be calculated by summing the length of all not-overwritten containee instances, i.e.

$$len_{i,j}^{P} = \text{length}_i^{P-\text{H}} + \sum_{\substack{\ell \in \text{PDU}_i^P, \\ k \in \Omega_\ell^P}} (1 - o_{i,j,\ell,k}^{P}) \cdot c_{i,j,\ell,k}^{P} \cdot len_{i,j}^{P} \tag{6}$$

for all PDUs i, ℓ and $j \in \Omega_i^P$, $k \in \Omega_\ell^P$ where $\text{length}_i^{P-\text{H}}$ is the length of the header. In order to ensure correct modeling of the collection semantics, the following constraints are added conditionally:

$$n_{i,j} \leq n_{i,j+1} \qquad \text{if } i \text{ is collected } \textit{last-is-best} \tag{7}$$

$$n_{i,j} < n_{i,j+1} \qquad \text{if } i \text{ is collected } \textit{queued} \tag{8}$$

To determine the possible point of time for the triggering of a container PDU, the minimum of the values is used:

$$\alpha_{i,j}^{P} = \min\{\, \alpha_{i,j}^{P-C1}, \alpha_{i,j}^{P-CT}, \alpha_{i,j}^{P-Cn} \,\}. \tag{9}$$

If one of the triggers is not applicable, the respective value is set to T_{sup}. This means, if $\alpha_{i,j}^{P} = T_{\text{sup}}$ the instance of the container PDU has not been triggered and must not be considered further if not triggered

due to a timeout. Finally, the triggering of an instance j of an PDU i happens at the first point of time it is possibly triggered by any of the described triggers, i.e.

$$\alpha^P_{i,j} = \min\{\, \alpha^{P-E}_{i,j}, \alpha^{P-T}_{i,j} \,\} \tag{10}$$

for all PDUs i and PDU occurrences $j \in \Omega^P_i$.

Given $\alpha^P_{i,j}$ it can be described in which occurrences of the encapsulating PDU (PDU^P) j is possibly transmitted. However, since we are considering container PDUs with dynamic layouts decided at runtime, a PDU is not necessarily send within the next instance of a container. In other words, if the container PDU is already filled to capacity, the containee has to wait until the next instance is sent. To model this, we constrain all instances of the container which are sent between the instance encapsulating the an instance j of an PDU i and the point of time j was triggered to be too full to contain j. The variable $n^P_{i,j}$ holds index of the encapsulating instance for all PDUs i and $j \in \Omega^P_i$. The variable $fn^P_{i,j}$ holds the index of the first instance which is a candidate for encapsulation, i.e. for all PDUs i and PDU occurrences $j \in \Omega^P_i$ which are mapped to a container PDU ℓ,

$$fn^P_{i,j} = \min\left(\{\,\text{occ\_max}^P_\ell\,\} \cup \{\, k \mid k \in \Omega^P_\ell \wedge \alpha^P_{\ell,k} \geq \alpha^P_{i,j} \,\}\right). \tag{11}$$

If the collection semantic of the PDU i into the container PDU ℓ is *queued*, the following constraint is added for each occurrence j in $j \in \Omega^P_i$:

$$\forall k \in \Omega^P_\ell \colon (k \geq fn^P_{i,j} \wedge k > n^P_{i,j-1} \wedge k < n^P_{i,j}]) \rightarrow (len^P_{\ell,k} + len^P_{i,j} > \text{length}^P_\ell). \tag{12}$$

Otherwise, if the collection semantic of the PDU i into the container PDU ℓ is *last-is-best*, the constraints for j in $j \in \Omega^P_i$ are depending on whether an instance of the same PDU is already part of the container. If this is the case, the content would simply be overwritten. Otherwise, if there exists no $j' \in \Omega^P_i \setminus \{\, j \,\}$ such that $n^P_{i,j'} = k$, the following constraint needs to be added:

$$\forall k \in \Omega^P_\ell \colon (k \geq fn^P_{i,j} \wedge k < n^P_{i,j}) \rightarrow (len^P_{\ell,k} + len^P_{i,j} > \text{length}^P_\ell). \tag{13}$$

It is important to note that in this case, instances of PDU i must be excluded when computing the length of ℓ as previously contained instances would be overwritten.

Together with the constraint for the length of a container PDU, the Constraints 12 and 13 assure that if no non-full container can be sent if one of its contained PDU is queued for sending and could fit into the container instance lengthwise.

Once triggered, the frame is queued for transmission on the bus. Thereupon, the frame data is copied from global memory to the buffer of the communication controller by the next instance of the responsible communication task. Therefore, in order to get the relative temporal distance between sending and receiving communication task, the possible point of time it was queued for transmission needs to be tracked for an instance of a frame. The following constraint is added for all frames i and their instances $j \in \Omega^F_i$:

$$\alpha^F_{i,j} = \min\{\, \alpha^P_{\ell,k} \mid \ell \in \text{PDU}^F_i, k \in \Omega^P_\ell \wedge n^P_{\ell,k} = j \,\}. \tag{14}$$

In the following, we describe the constraints needed to model CAN FD network. They can easily be plugged in by adding the corresponding constraints. The constraints listed up to this point can consequently be reused for other bus types. Let iTx be the communication task for frame i and jTx be the first instance of this iTx after occurrence j of i was queued for transmission. Furthermore, let $\alpha^T_{iTx,jTx}$

be the activation time of this task and frame_i^{HP} the set of frames which have a higher priority than i. The following constraint is added to model the start of transmission:

$$\sigma_{i,j}^F = \max\{\alpha_{iTx,jTx}^T\} \cup \{\varepsilon_{\ell,k}^F \mid \ell \in \text{frame}_i^{HP}, k \in \Omega_\ell^F \wedge \sigma_{\ell,k}^F \leq \sigma_{i,j}^F\}. \tag{15}$$

For CAN FD, the transmission time of a frame is resulting from an arbitration phase and the time the bits are transmitted on the physical medium. In the arbitration phase, all nodes of the network agree on the node allowed to transmit next. The node trying to transmit the frame with the highest priority is allowed to continue sending after arbitration. From Constraint 15 it can be inferred that the occurrence j of frame i has the highest priority at point of time $\sigma_{i,j}^F$. Therefore, the end of transmission $\varepsilon_{i,j}^F$ can be calculated in the following way:

$$\left\lfloor t_{\text{arb}} + (len_{i,j}^F \cdot t_{\text{bit}}) \right\rfloor \leq \varepsilon_{i,j}^F \leq \left\lceil t_{\text{arb}} + (len_{i,j}^F \cdot t_{\text{bit}}) \right\rceil \tag{16}$$

where t_{arb} worst-case time needed for arbitration and t_{bit} is the time needed to transfer one bit of data. The length of the instance j of frame i, $len_{i,j}^F$, can be computed analogously to the PDUs (cf. Constraint 6). With the help of $\varepsilon_{i,j}^F$ the instance of the communication task at the receiving ECU can be determined. The time between the initial change of the signal and the deadline of this task is the time needed for a value change of this signal to be transferred from the global memory of one ECU to the next. Depending on whether the clocks of the ECUs in the network are synchronized, a clock drift can additionally be considered for the receiving communication task.

Given all constraints for a set of network artifacts, a constraint solver is deployed to obtain the worst-case transmission time. The solver searches all satisfying assignments for the one modeling the longest transmission time. This emulates an exhaustive search if all situations are covered. To guarantee this, all possible relative offset between the PDU containing the objective signal and all PDUs possibly causing a delay of this PDU needs to be covered. If all relevant PDUs are triggered by periodic timers or periodically changing signals, the least common multiple of these periods covers all relative offsets. If sporadic events are included, a lot more relative offsets are possible. However, in this case the solver freely chooses values between the first and last possible point of time for the event to happen (cf. Constraint 1). For practical application, it is furthermore important that $\mathbb{T}$ covers a sufficiently large time interval such that it is possible that all PDUs which are triggered can also be sent.

4.3 Estimations for Software Task Chains

Software task chains can either start with a sensor reading or a network communication task and end at an actuator or another network communication task. Regardless of type, estimating latencies for task chains is concerned with the question which relative offsets are possible between task instances on the chain. The activation pattern of the task, the method for accessing memory, as well as the core execution times are factors affecting the possibilities for these relative offsets. In Section 2 state-of-the art approaches to obtain latencies for different kind of task chains are listed. We want to highlight the approach described in [12] as it provides the interesting possibility to create a combined model for both types of sub-chains considered in this work. The combination of both approaches would allow for an end-to-end timing model comprising multiple ECUs and networks. Although attention needs to be payed to potential scalability problems such a model is promising in terms of the precision of the results.

4.4 End-to-end Estimations

Due to complexity, estimations of end-to-end latencies are often performed on smaller, easier to handle segments of a chain. This comes with the price of a *context loss* at each *cut* of the analysis. After a

context loss, most information on the situation which lead to the worst case in previous segment need to be dropped. As a result, mutually exclusive situations might be considered cumulatively. This can lead to significant overestimations. To reduce the impact of context losses, we propose to cut analysis only at communication tasks. This cut essentially results in two kind of sub chains: communication task to communication task via software and via network. An advantage of cutting analysis at communication tasks is, that context losses with regard to changes of signal values are not too costly in terms of information loss. The task chain on the next ECU certainly starts with the receiving communication task where local signals of the sending ECU have no direct impact. Moreover, having a context loss prior to determining the possibilities for the PDU trigger is unavoidable with many approaches to estimate task chains. They only keep track of the one signal relevant for the chain. However, as described above, different signals can be decisive for the triggering of the PDU transporting the signal over the network. With the approach presented in this work, the trigger mechanisms for PDUs can be considered independently.

5 Experiments

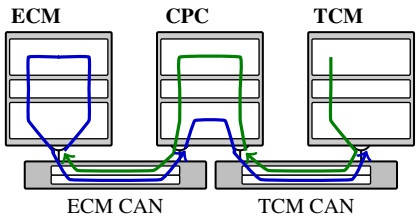

Figure 2: Case-Study: Topology and Chain

To test the constraint model for the estimation of signal transmission times, we applied it on a case study based on real automotive data. It contains synthetic but realistic data for two CAN-FD networks connecting three ECUs as depicted in Figure 2. The amount of frames and PDUs roughly matches the amount of artifacts for the functional communication between the powertrain domain controller (CPC), the engine controller (ECM), and the transmission controller (TCM). Responses to diagnostic messages as well as network management communication is not considered. Furthermore, only signals potentially having an impact on the triggering of a PDU are included. Two signals are aggregated to one signal if it has no impact on the estimation. More precisely, two signals are combined to one modeling the same behavior if they are sent as part of the same PDU with the same trigger-related properties and periodic timings and neither is the objective signals. In this way, the amount of signals in the model is greatly reduced when compared to a naive model.

We analyzed the worst-case time for the transmission of eight different signals, four contained in dynamic container PDUs, four being part of the cause-effect chain depicted in Figure 2. The first-mentioned signals represent status updates sent from the ECM and TCM to the CPC. The cause-effect chain on the other hand is a control loop and has two parts. The first part of the chain describes the data flow when the TCM sends a request for torque to the ECM. The CPC checks the requests before forwarding it to the ECM. After providing torque to the extend possible, the ECM reports to the CPC. The answer of the ECM is directly routed from the ECM CAN to the TCM CAN for fast reception at the TCM.

To implement the constraint model described in Section 4 we used the high-level constraint language MiniZinc. The constraints can be translated almost directly. An advantage of using MiniZinc is the descriptive nature of the resulting model. It is readable and verifiable for anyone with a basic understanding of constraint modeling. The greatest benefit however is the form of the results. If a feasible solution was found, constraint solvers do not only answer *yes* or *no* but they give a assignment for all variables of the constraint set. The feasible solutions for our set of time and decision variables can directly be interpreted to identify the situation which lead to the worst case. Each set of artifacts is encoded in a data file to

Table 2: Resource Usage for Compiling and Solving the Model

Sender	Receiver	Signal	Compiler		Solver			Result
			s	kbyte	s	kbyte		μs
ECM	CPC	Status A	2:29	4,435,568	2:30	57,462,088	*	15500
ECM	CPC	Status B	2:28	4,441,504	2:16	60,797,024	*	20500
TCM	CPC	Status C	0:55	1,723,920	0:46	8,772,148		15500
TCM	CPC	Status D	0:52	1,723,896	1:03	9,863,388		15500
TCM	CPC	Request 1	0:52	1,723,808	0:41	8,499,072		11250
CPC	ECM	Request 2	2:24	4,432,880	3:02	19,298,896		12500
ECM	CPC	Response 1	2:27	4,466,500	2:45	91,870,500		5500
CPC	TCM	Response 2	0:56	1,734,188	0:49	9,050,628		6000

*Running instances of the solver were interrupted after the optimum was found.

replace constraint parameters with their actual values. This data file is then linked to the constraints and compiled to a second constraint model in the low-level constraint language FlatZinc. FlatZinc is supported by different solver back-ends. We used the parallel version of the lazy-clause generation constraint solver *Chuffed* [9]. The experiments have been carried out on a desktop computer equipped with an Intel(R) Core(TM) i9-7940X CPU and 128GB of memory. The time and memory needed to compile the MiniZinc model as well as the time and memory needed for solving the FlatZinc model by 16 instances of *Chuffed* working in parallel are shown in Table 2.

6 Future Work

In future work we want to examine the possibilities arising from the integration of software-level and network-level models. The scalability of a holistic model must be examined. If it can be contained by with current hardware and solver technologies, it could not only be used to verify system designs but also to automatically generate suggestions on the optimization. In distributed system with cause-effect chains spanning over multiple ECUs the effects of changes are very difficult to predict and might influence the performance of other cause-effect chains. With precise end-to-end estimations however, these impacts could be estimated quickly to evaluate system designs quickly.

7 Conclusion

In this paper we addressed the problem of estimating the end-to-end latency of distributed cause-effect chains in automotive systems. A previously unsupported part of the problem is estimating the temporal behavior of the PDU triggering mechanism. For this, we introduced timing models which can be combined to model the temporal behavior of two or more event sources. Based on this we presented a constraint model to estimate the time needed to transmit signal changes in CAN FD communication clusters. We discussed how this approach can be integrated in an end-to-end analysis and why this would improve estimations. Finally, the application on an OEM use case shows scalability for industrial-scale problems.

References

[1] AUTomotive Open System ARchitecture (2017): *Specification of I-PDU Multiplexer*. Available at `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_IPDUMultiplexer.pdf`.

[2] AUTomotive Open System ARchitecture (2017): *Specification of RTE Software*. Available at `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_RTE.pdf`.

[3] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam & Thomas Nolte (2016): *Synthesizing Job-Level Dependencies for Automotive Multi-rate Effect Chains*. In: *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2016, Daegu, South Korea, August 17-19, 2016*, IEEE Computer Society, pp. 159–169, doi:10.1109/RTCSA.2016.41.

[4] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam & Thomas Nolte (2017): *End-to-end timing analysis of cause-effect chains in automotive embedded systems*. *J. Syst. Archit.* 80, pp. 104–113, doi:10.1016/j.sysarc.2017.09.004.

[5] Frédéric Boniol, Michaël Lauer, Claire Pagetti & Jérôme Ermont (2013): *Freshness and Reactivity Analysis in Globally Asynchronous Locally Time-Triggered Systems*. In Guillaume Brat, Neha Rungta & Arnaud Venet, editors: *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, *Lecture Notes in Computer Science* 7871, Springer, pp. 93–107, doi:10.1007/978-3-642-38088-4_7.

[6] Jean-Yves Le Boudec & Patrick Thiran (2001): *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. *Lecture Notes in Computer Science* 2050, Springer, doi:10.1007/3-540-45318-0.

[7] Rene L. Cruz (1991): *A calculus for network delay, Part I: Network elements in isolation*. *IEEE Trans. Information Theory* 37(1), pp. 114–131, doi:10.1109/18.61109.

[8] Robert I. Davis, Alan Burns, Reinder J. Bril & Johan J. Lukkien (2007): *Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised*. *Real-Time Systems* 35(3), pp. 239–272, doi:10.1007/s11241-007-9012-7.

[9] Thorsten Ehlers & Peter J. Stuckey (2016): *Parallelizing Constraint Programming with Learning*. In Claude-Guy Quimper, editor: *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings*, *Lecture Notes in Computer Science* 9676, Springer, pp. 142–158, doi:10.1007/978-3-319-33954-2_11.

[10] Rolf Ernst, Leonie Ahrendts & Kai Björn Gemlau (2018): *System Level LET: Mastering Cause-Effect Chains in Distributed Systems*. In: *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society, Washington, DC, USA, October 21-23, 2018*, IEEE, pp. 4084–4089, doi:10.1109/IECON.2018.8591550. Available at `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8560606`.

[11] Nico Feiertag, Kai Richter, Johan Nordlander & Jan Jonsson (2008): *A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics*. In: *Proceedings of the IEEE Real-Time System Symposium - Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, Barcelona, Spain, November 30, 2008*. Available at `http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-30306`.

[12] Max J. Friese, Thorsten Ehlers & Dirk Nowotka (2018): *Estimating Latencies of Task Sequences in Multi-Core Automotive ECUs*. In: *13th IEEE International Symposium on Industrial Embedded Systems, SIES 2018, Graz, Austria, June 6-8, 2018*, IEEE, pp. 1–10, doi:10.1109/SIES.2018.8442095.

[13] Kai-Björn Gemlau, Johannes Schlatow, Mischa Möstl & Rolf Ernst (2017): *Compositional Analysis for the WATERS Industrial Challenge 2017*. In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Dubrovnik, Croatia.

[14] Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia & Nicolas Sordon (2018): *Improving and Estimating the Precision of Bounds on the Worst-Case Latency of Task Chains*. *IEEE Trans. on CAD of Integrated Circuits and Systems* 37(11), pp. 2578–2589, doi:10.1109/TCAD.2018.2861016.

[15] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler & Falk Wurst (2017): *Communication Centric Design in Complex Automotive Embedded Systems*. In Marko Bertogna, editor: *29th Euromicro Conference on Real-Time Systems, ECRTS 2017, June 27-30, 2017, Dubrovnik, Croatia, LIPIcs* 76, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 10:1–10:20, doi:10.4230/LIPIcs.ECRTS.2017.10.

[16] Thomas A. Henzinger, Benjamin Horowitz & Christoph M. Kirsch (2001): *Giotto: A Time-Triggered Language for Embedded Programming*. In Thomas A. Henzinger & Christoph M. Kirsch, editors: *Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings, Lecture Notes in Computer Science* 2211, Springer, pp. 166–184, doi:10.1007/3-540-45449-7_12.

[17] Shugang Jiang (2019): *Vehicle E/E Architecture and Its Adaptation to New Technical Trends*. In: *WCX SAE World Congress Experience*, SAE International, doi:10.4271/2019-01-0862.

[18] Leonie Köhler, Borislav Nikolic, Rolf Ernst & Marc Boyer (2019): *Increasing Accuracy of Timing Models: From CPA to CPA+*. In Jürgen Teich & Franco Fummi, editors: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, IEEE, pp. 1210–1215, doi:10.23919/DATE.2019.8714770. Available at `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8704855`.

[19] Rodrigo Lange, Alexsandro C. Bonatto, Francisco Vasques & Rômulo Silva de Oliveira (2016): *Timing Analysis of hybrid FlexRay, CAN-FD and CAN vehicular networks*. In: *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, Italy, October 23-26, 2016*, IEEE, pp. 4725–4730, doi:10.1109/IECON.2016.7793791. Available at `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7782522`.

[20] Rodrigo Lange, Rômulo Silva de Oliveira & Francisco Vasques (2016): *A reference model for the timing analysis of heterogeneous automotive networks*. *Computer Standards & Interfaces* 45, pp. 13–25, doi:10.1016/j.csi.2015.10.004.

[21] Michaël Lauer, Frédéric Boniol, Claire Pagetti & Jérôme Ermont (2014): *End-to-end latency and temporal consistency analysis in networked real-time systems*. *IJCCBS* 5(3/4), pp. 172–196, doi:10.1504/IJCCBS.2014.064667.

[22] Edward A. Lee (2006): *Cyber-Physical Systems - Are Computing Foundations Adequate? Proc. Workshop CyberPhysical Syst. Res. Motiv. Tech. Roadmap.* Available at `http://ptolemy.eecs.berkeley.edu/publications/papers/06/CPSPositionPaper/`.

[23] Jorge Martinez, Ignacio Sanudo Olmedo & Marko Bertogna (2018): *Analytical Characterization of End-to-End Communication Delays With Logical Execution Time*. *IEEE Trans. on CAD of Integrated Circuits and Systems* 37(11), pp. 2244–2254, doi:10.1109/TCAD.2018.2857398.

[24] Moritz Neukirchner, Mircea Negrean, Rolf Ernst & Torsten T. Bone (2012): *Response-time analysis of the flexray dynamic segment under consideration of slot-multiplexing*. In: *7th IEEE International Symposium on Industrial Embedded Systems, SIES 2012, Karlsruhe, Germany, June 20-22, 2012*, IEEE, pp. 21–30, doi:10.1109/SIES.2012.6356566. Available at `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6339458`.

[25] Kai Richter, Dirk Ziegenbein, Marek Jersak & Rolf Ernst (2002): *Model composition for scheduling analysis in platform design*. In: *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*, ACM, pp. 287–292, doi:10.1145/513918.513993.

[26] Johannes Schlatow & Rolf Ernst (2016): *Response-Time Analysis for Task Chains in Communicating Threads*. In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*, IEEE Computer Society, pp. 245–254, doi:10.1109/RTAS.2016.7461359. Available at `http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7460013`.

[27] Simon Schliecker, Jonas Rox, Mircea Negrean, Kai Richter, Marek Jersak & Rolf Ernst (2009): *System Level Performance Analysis for Real-Time Automotive Multicore and Network Architectures*. *IEEE Trans. on CAD of Integrated Circuits and Systems* 28(7), pp. 979–992, doi:10.1109/TCAD.2009.2013286.

[28] Daniel Thiele, Johannes Schlatow, Philip Axer & Rolf Ernst (2016): *Formal timing analysis of CAN-to-Ethernet gateway strategies in automotive networks*. Real-Time Systems 52(1), pp. 88–112, doi:10.1007/s11241-015-9243-y.

[29] Lothar Thiele, S. Chakraborty & M. Naedele (2000): *Real-time calculus for scheduling hard real-time systems*. In: *IEEE International Symposium on Circuits and Systems, ISCAS 2000, Emerging Technologies for the 21st Century, Geneva, Switzerland, 28-31 May 2000, Proceedings*, IEEE, pp. 101–104, doi:10.1109/ISCAS.2000.858698. Available at `http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=18601`.