Real-Time Syst (2016) 52:88–112
DOI 10.1007/s11241-015-9243-y

Formal timing analysis of CAN-to-Ethernet gateway strategies in automotive networks

Daniel Thiele[1] · **Johannes Schlatow**[1] · **Philip Axer**[1] · **Rolf Ernst**[1]

Published online: 7 October 2015

Abstract Due to increased bandwidth and scalability demands, Ethernet technology is finding its way into recent in-vehicle networks. Tomorrow's heterogeneous networks will feature legacy buses [e.g. controller area network (CAN) or FlexRay] as well as high-speed Ethernet devices, connected by switches and gateways. As Ethernet offers significantly larger frame sizes than CAN, the efficient transmission of CAN data over an Ethernet backbone depends heavily on the way this data is multiplexed into Ethernet frames. This article focuses on the timing impact introduced by various CAN/Ethernet multiplexing strategies at the gateways. We present a formal analysis method to derive upper bounds on end-to-end latencies for complex multiplexing strategies, which is key for the design of safety-critical real-time systems. We capture complex inter-domain signal paths spanning multiple buses, gateways, and switches and show the applicability in a realistic automotive setup.

Keywords Automotive · Ethernet · Real-time · Embedded systems · Performance analysis · CAN · Gateway · Networks · Distributed systems

✉ Daniel Thiele
thiele@ida.ing.tu-bs.de

Johannes Schlatow
schlatow@ida.ing.tu-bs.de

Philip Axer
axer@ida.ing.tu-bs.de

Rolf Ernst
ernst@ida.ing.tu-bs.de

1 Institute of Computer and Network Engineering, Technische Universität Braunschweig, Hans-Sommer-Str. 66, 38106 Braunschweig, Germany

1 Introduction

The automotive network backbone, today built up on controller area network (CAN), CAN-FD (flexible data rate), or FlexRay, reaches its limits in bandwidth and scalability. New advanced driver assistance systems add an extraordinary amount of streaming data and additional control signals which cannot be handled by today's network infrastructure. Many automotive manufacturers therefore consider switched Ethernet to be used as a potential high-speed backbone, as an overlay network or, in long term perspective, even as a full replacement for most legacy buses.

In the near future, a transition phase is expected, where an Ethernet network will serve as the backbone to interconnect various vehicle domains (cf. Fig. 1). Low-speed buses (LIN, CAN) will still be used due to their competitive price and legacy compatibility. High-speed devices such as cameras, on the other hand, will be connected directly to Ethernet switches. The figure does not depict the actual Ethernet topology. A daisy-chain, a star, or a tree of arbitrary shape can be implemented, depending on bandwidth requirements and the available number of ports per switch.

One of the major challenges in automotive network design is the *timing determinism* of critical control and streaming data (Zhang et al. 2013). It is crucial that sensor and actuator data are transmitted within well-specified latency bounds to meet the constraints of time-critical driving functions and prevent controller degradation. This is especially important for safety-critical applications which put human lives at stake. Formal timing analysis, known for example from Real-Time Calculus (RTC) (Thiele et al. 2000) and Compositional Performance Analysis (CPA) (Henia et al. 2005), assists engineers to formally prove timing constraints such as network end-to-end latencies and gateway response times.

In this article we focus on the formal end-to-end timing analysis of inter-domain traffic with real-time requirements. As an example, we use traffic which is being sent over an Ethernet backbone from a CAN bus in one domain to a CAN bus in a different domain (e.g. Powertrain to Body in Fig. 1). To transport CAN frames (i.e. their ID and signal payload) over an Ethernet network, there are two extrema: (a) sending each CAN frame via a dedicated Ethernet frame and (b) multiplexing (packing) as many CAN frames as possible into an Ethernet frame. While the first approach intuitively results in low latencies (each CAN frame is sent immediately), it has a large Ethernet protocol

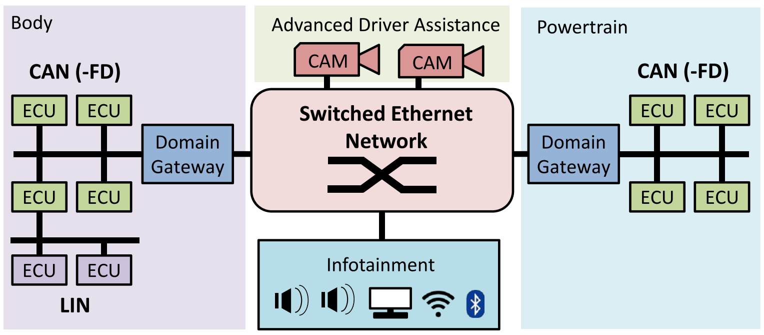

Fig. 1 Ethernet backbone with gateways and legacy buses

overhead ($\sim$90 %) and introduces a high load on the domain gateways and the Ethernet (packing and sending many small frames). The latter approach has minimal protocol overhead and might also allow the exploitation of Ethernet multicast. However, it suffers from potentially large latencies as Ethernet frames are only sent when they are full, which might take some time. An optimal solution is somewhere between these extrema and highly depends on the traffic characteristics.

While, in this article, we focus on strict-priority scheduled Ethernet (IEEE 802.1Q), our analysis methodology is scheduling agnostic and works seamlessly with Ethernet Audio/Video Bridging (AVB) or the upcoming Time Sensitive Networking (TSN) standards. As we will see in the evaluation, Ethernet has a negligible impact on the end-to-end latency. It is more important to focus on an efficient multiplexing strategy.

The main **contributions** of this article are formal methods to analyze the impact of different multiplexing strategies at the gateways on the end-to-end latency guarantees. Specifically, we focus on three different aspects of multiplexing: (a) how CAN frames are multiplexed and demultiplexed (buffering), (b) which CAN frames are multiplexed together (mapping), and (c) when an Ethernet frame is sent (triggering). Our analysis method is able to capture and quantify the (significant) differences between different multiplexing strategies.

The article is structured in the following way: First, we discuss related work in the field of system performance analysis and gateway analysis in Sect. 2. In Sect. 3, we present an overview of the CPA methodology and, in Sect. 4, we show how it can be applied to automotive networks. We extend the CPA methodology to cover the peculiarities of multiplexing CAN traffic at the gateways in Sect. 5. The applicability of our proposed approach is evaluated in Sect. 6. Finally, we conclude the article in Sect. 7.

2 Related work

The investigation of the timing impact of gateway strategies in heterogeneous automotive communication systems requires a multitude of analysis methods, each specialized to a specific part of the communication path. CAN buses and gateways typically utilize static-priority scheduling (non-preemptive and preemptive, respectively), which can be analyzed by the busy-period approach (e.g. Lehoczky 1990, Davis et al. 2007). Here, the worst-case response time is derived by constructing the worst-case time during which a resource is busy serving requests at a given priority.

There exists a large body of related work in timing analysis for switched networks. Most of the work uses either RTC (Thiele et al. 2000) or CPA (Henia et al. 2005). RTC has been used for Ethernet analysis in e.g. Revsbech et al. (2011). Formal timing analysis using CPA for strict-priority Ethernet was presented in Rox and Ernst (2010). In Bauer et al. (2010) the trajectory approach was applied to analyze Avionics Full Duplex Switched Ethernet (AFDX).

Different CAN to Ethernet multiplexing strategies for gateways have been studied by Kern et al. (2011), Ayed et al. (2011), Scharbarg et al. (2005), Nacer et al. (2013), and Herber et al. (2015). In Kern et al. (2011) a prototypical CAN/Ethernet multiplexing gateway has been implemented to generate experimental results for average end-to-end latencies, required network bandwidth, and resource utilization of the gateway processor for different multiplexing strategies. The authors of Ayed et al. (2011) present two simple multiplexing strategies (a 1:1 mapping and a timeout-based strategy) along with a formal analysis and propose an optimization of the timeout value regarding schedulability and gateway resources. Scharbarg et al. (2005) and Nacer et al. (2013) propose 1:1 mapping, buffer-full, and buffer-full with timeout multiplexing strategies. The evaluation of the multiplexing process, however, is only simulation based. In Herber et al. (2015) different buffering mechanisms for CAN frames at gateways are evaluated (i.e. which CAN frames are to be multiplexed first) under timeout-based triggering: FIFO, strict-priority, and earliest deadline first.

We, in contrast, model multiplexing as operations on event models (Henia et al. 2005), which can later be used seamlessly by a formal performance analysis, considering more complex triggering strategies [including the aforementioned ones and new strategies specified by AUTOSAR (2015)] and their combination. This allows us to formally derive the worst-case timing behavior on the Ethernet and the gateways, which, as we will see, has a significant impact on end-to-end timing. The importance of this work is stressed by the fact that the AUTOSAR Socket Adapter Specification (AUTOSAR 2015) already specifies multiplexed transmissions of payload data units via UDP and TCP sockets. In contrast to Herber et al. (2015), we only consider FIFO buffering and, instead, assume that time-critical CAN frames can trigger the immediate transmission of an Ethernet frame (as proposed by AUTOSAR 2015), eliminating the requirement of complex frame ordering strategies at the gateways, while also eliminating the additional delay of waiting for the next timeout to occur. Additionally, our approach supports lossy buffering, i.e. old values are overwritten by newer ones.

The multiplexing of individual control signals into (CAN) frames is discussed in e.g. Pop et al. (2005) and Saket and Navert (2006). While related to the multiplexing of CAN frames into Ethernet frames, the authors do not focus on the derivation of formal event models, but instead suggest heuristics to solve the mapping problem (which signal is packed into which frame) during multiplexing. Thus, these works are orthogonal to our approach.

Furthermore, there is work on the (hierarchical) composition and decomposition of event streams in join-fork scenarios that are applicable for the timing analysis of inter-gateway communication. In Rox and Ernst (2008), the construction and deconstruction (multiplexing and demultiplexing) of Hierarchical Event Streams (HES) was formally defined for AND-/OR-join semantics with timeout and trigger events. It also defines update functions for the modification of jitter and minimum distance of the inner event streams of an HES. Similarly, Structured Event Streams, which are characterized by so-called Event Count Curves, were introduced in Perathoner et al. (2010) and compared against HES.

3 Compositional performance analysis

In CPA (Henia et al. 2005), systems are modeled by sets of resources and tasks. A resource provides service (processing or transmission time) which is consumed by the tasks mapped to it. Contention for resources with multiple tasks is resolved according to the resource's scheduling policy (e.g. static-priority preemptive).

The execution behavior of a task τ_i is divided into the following steps: activation, core execution, and finally completion/propagation. After being activated, a task (or job) is ready to execute and can be scheduled. Depending on the scheduling policy and resource type, tasks may be interrupted by other tasks running on the same resource. Each invocation of a task imposes a bounded workload to the resource, i.e. its core execution time, which is bounded by the worst-case and best-case execution times denoted C_i^+ and C_i^-, respectively.

A distributed application consisting of multiple communicating tasks is described by a directed graph in which nodes are tasks and edges represent functional data dependencies. After a task's execution is completed, the task activates its dependent tasks (propagation). Here, forks are possible, i.e. one task can activate multiple other tasks. The opposite, i.e. a join, requires an explicit semantic. There are two common join-semantics as discussed by Henia et al. (2005): For an *OR-join* any incoming event produces one outgoing event, and for an *AND-join*, an outgoing event is produced once events are available on all incoming edges.

In CPA, the communication between tasks is abstracted by an *event model interface*. As discussed above, task activation and completion denote specific events of interest. Events can also originate from external sources, such as a timer or external devices. During the analysis, the actual data is of no further interest but rather the points in time when events occur is modeled (e.g. times when data is communicated). To keep the analysis problem tractable, an event model interface abstracts the actual event trace by only capturing worst-case and best-case behavior. An event model consists of a pair of arrival curves $\eta_i^-(\Delta t)/\eta_i^+(\Delta t)$, which return a lower/upper bound on the number of events that can arrive within any half-open time window $[t, t + \Delta t)$ Richter (2005). Thus, any trace that is between these bounds is covered by the event model representation. These functions have pseudo-inverse counterparts, the so-called maximum/minimum-distance functions $\delta_i^+(n)/\delta_i^-(n)$, which return an upper/lower bound on the time interval between the first and the last event of any sequence of n event arrivals. For convenience we introduce the event model interfaces $\boldsymbol{\delta}_i = \{\delta_i^-, \delta_i^+\}$ and $\boldsymbol{\eta}_i = \{\eta_i^+, \eta_i^-\}$ to refer to an event stream's worst-case and best-case event models. Often, it is more convenient to consider δ functions, due to their discrete domain. A conversion between $\boldsymbol{\delta}_i$ and $\boldsymbol{\eta}_i$ is described by Diemer et al. (2012a).

The CPA analysis flow, as shown in Fig. 2, consists of two interleaved steps: the *local analysis* and the *propagation* of event models. The environment model specifies

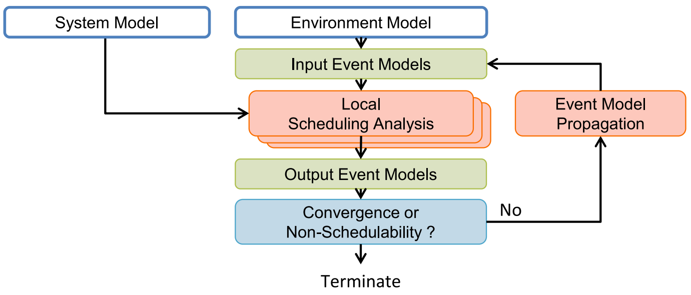

Fig. 2 Compositional analysis flow

the boundary conditions for each event stream under which the system is operating. This is usually given as external event models which describe the characteristics of external sources. Other, yet unknown, event models are initialized with optimistic guesses that are iteratively updated during analysis. The event models are used for the local resource analysis during which the local behavior of each resource is considered in isolation. Among other results, the local analysis yields the response-time jitter from which new output event models can be derived (Henia et al. 2005). These output event models then become the input event models of dependent task.

The iteration is stopped and a system fixed-point is found if all event models remain stable. Then, the resulting best-case/worst-case response times (R_i^- / R_i^+) along a path can be added up to retrieve end-to-end path latencies L_i^- / L_i^+. Similarly, if we are interested in the delay of multiple activations, we can use the following equation (Diemer et al. 2011):

$$L_i^{-/+}(q) = \delta_i^-(q) + \sum_{j \in Path(i)} R_j^{-/+} \tag{1}$$

Here, $Path(i)$ is the set of all tasks which belong to a path i. Equation 1 assumes a maximum injection rate at the event source, i.e. q events are generated within a time interval of $\delta_i^-(q)$. This will cause the system's resources to exhibit worst-case behavior such as the worst-case response times R_i^+. The general idea behind Eq. 1 is that the last of the q events will experience the worst-case on each resource. Due to causality, all $q - 1$ previously sent events must have arrived by then.

4 Modeling automotive networks

In this section, we discuss how a heterogeneous automotive network, consisting of multiple ECUs, buses, Ethernet switches, and gateways is modeled using CPA primitives. An exemplary architecture is shown in the upper part of Fig. 3 and the corresponding logical CPA model in the lower part. Two CAN buses are connected to an Ethernet switch via gateways. Individual communication streams are indicated by colored arrows in the upper part of the figure. One ECU is directly connected to the Ethernet switch. In this particular example, CAN frames available on CAN1 are sent to CAN2 (red arrow) and from CAN2 to CAN1 (yellow arrow). Some CAN frames from CAN1 are sent via multicast to the ECU for further processing and the result is sent back to CAN1.

The resulting CPA model is shown in the lower part of Fig. 3. Edges represent event model interfaces which are given as environment models $\boldsymbol{\delta}_{1,in}$ to $\boldsymbol{\delta}_{5,in}$ or are derived during analysis. Multicast traffic (CAN1 sends data to CAN2 and ECU) is modeled by forking the event stream. Most physical resources are directly modeled as CPA resources (rectangles with black name tags). An exception is the Ethernet switch, which is modeled as described by Diemer et al. (2012b). Each output port, as a point of arbitration, is modeled as a resource. We assume that all other switch delays can be modeled by constant delays Diemer et al. (2012b). Software tasks are shown as white tasks. The transmission of CAN and Ethernet frames is depicted as grey tasks.

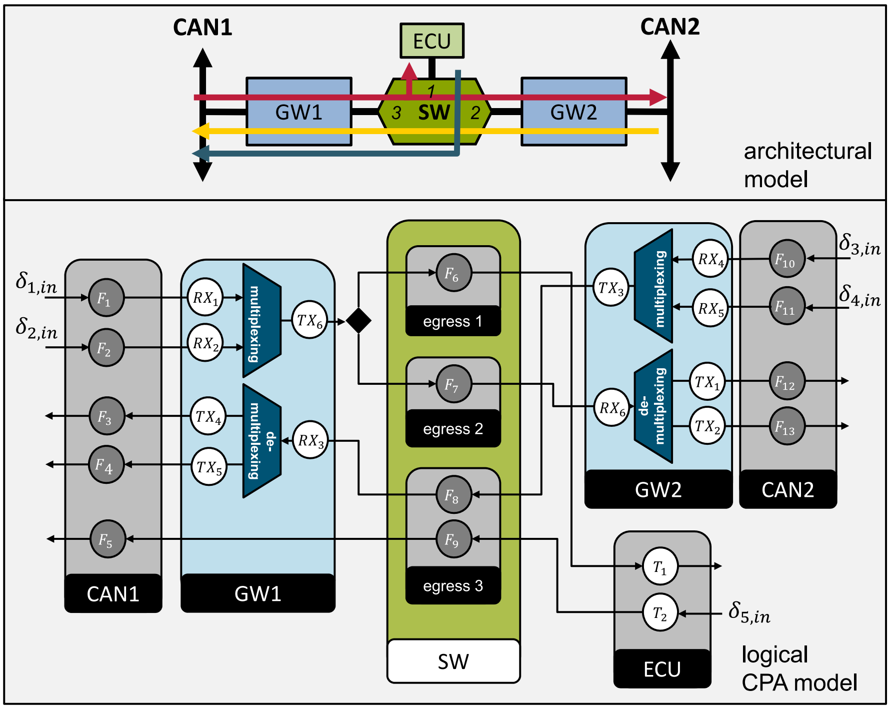

Fig. 3 System model, consisting of two buses, two gateways, an Ethernet switch connected to the gateways, and an ECU

Due to network heterogeneity, different CPA analyses must be utilized to derive end-to-end latency guarantees, e.g. from CAN1 to CAN2. For the analysis of the CAN buses, we resort to an analysis approach based on Davis et al. (2007). We assume that frames in the Ethernet network are scheduled according to IEEE 802.1Q, i.e. in a static-priority non-preemptive way (Rox and Ernst 2010; Thiele et al. 2014). A detailed discussion of the modeling and timing analysis of the gateways is given in the following section. All analyses are interconnected by event model interfaces.

5 Ethernet gateways

In this section, we present CPA techniques to model different frame multiplexing strategies, i.e. how CAN frames are buffered into an Ethernet frame, how CAN frames are mapped to Ethernet frames, when Ethernet frames are sent, as well as the demultiplexing of CAN frames from Ethernet frames.

5.1 CPA gateway model

Without loss of generality, we reduce the discussion to the unidirectional scenario in Fig. 4 in which we distinguish between the ingress and the egress gateway. The

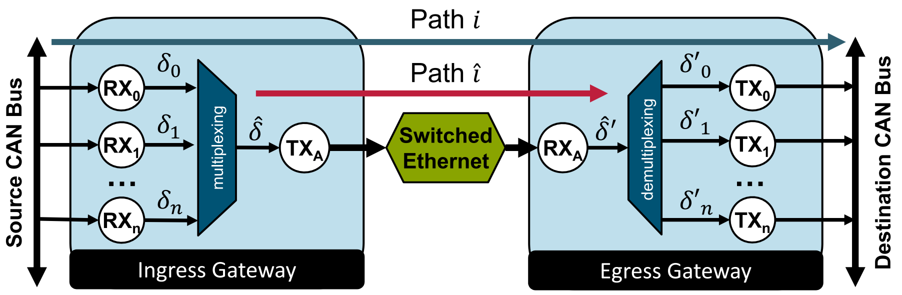

Fig. 4 An ingress gateway sends a multiplexed CAN traffic stream via a switched Ethernet to an egress gateway. The first merges in-event models, the latter demultiplexes the stream

ingress gateway receives CAN frames from one or multiple CAN buses and packs these frames into one or multiple Ethernet frames (multiplexing). The egress gateway receives Ethernet frames and unpacks the contained CAN frames in order to transmit them to the destination CAN bus(es) (demultiplexing). Bidirectional scenarios can be modeled by combining the task graphs of the ingress and egress gateways (cf. Fig. 3).

As we focus on CPA techniques to model different multiplexing strategies, we utilize simplified gateway models, which assume that any processing on the gateways can be split into a per-CAN-frame and a per-Ethernet-frame part. The ingress gateway comprises RX tasks for each incoming CAN frame as well as a TX task for each outgoing Ethernet frame. An RX task models the execution time that is required to process a CAN frame with a particular CAN identifier. Similarly, a TX task models the execution time of the Ethernet stack and the time that is required to transmit an Ethernet frame. The processor of the gateway is assumed to schedule tasks in static-priority preemptive (SPP) fashion (Schliecker et al. 2008). The TX task has the highest priority and RX tasks have a priority proportional to their CAN identifier below the TX task. Apart from the RX and TX tasks, a (multiplexing) join merges the event streams from multiple RX tasks to a single TX task. The semantics of this join junction depend on the multiplexing strategy and impact the *sampling delay* of the gateway (Feiertag et al. 2008). The sampling delay arises whenever multiple CAN frames are multiplexed into a single Ethernet frame during packetization, i.e. the first CAN frame in a new Ethernet frame must wait for the last CAN frame to be packed into the Ethernet frame.

The egress gateway is modeled by a task graph that basically reverses the operations of the ingress gateway. It contains an RX task for each incoming Ethernet frame and a TX task for each outgoing CAN frame. Here, the RX task has the highest priority and TX tasks have a priority proportional to their CAN identifier. In order to distribute a single activation from the RX task, which (in a multiplexing scenario) typically contains several CAN frames, to the TX tasks, a (demultiplexing) fork is required. As this fork, in contrast to the standard fork, which forwards all incoming event models to all dependent tasks, selectively distributes event models depending on the receiving task, we model the demultiplexing fork explicitly as in Perathoner et al. (2010). There is no sampling delay on the egress gateway.

5.2 Multiplexing strategies

We classify multiplexing strategies in terms of three fundamental parameters: *buffering*, *triggering* and *mapping*. *Buffering* takes place at the ingress gateway in order to collect incoming CAN frames. Once a certain condition is met (*triggering*), the contents of the buffer are sent as payload of an Ethernet frame. If multiple buffers are used, a *mapping* function determines to which buffer a CAN frame is forwarded.

We distinguish between lossy and lossless buffering, which fundamentally affect the timing analysis. A lossy strategy assumes that each CAN frame designated to a certain buffer has a fixed position in this buffer and will update (overwrite) the buffer content (or parts of it) with the most recent value and is suitable for transmitting sensor data. Conversely, a lossless strategy is order preserving and (non-destructively) queues data in the buffer.

Based on Kern et al. (2011), Ayed et al. (2011), Scharbarg et al. (2005), Nacer et al. (2013), and AUTOSAR (2015), Ethernet frames can be triggered by:

- *Buffer Timeout* This time-trigger is specified by a fixed timeout per buffer so that the gateway emits an Ethernet frame periodically (Kern et al. 2011; Ayed et al. 2011; Scharbarg et al. 2005; Nacer et al. 2013; AUTOSAR 2015).
- *Buffer-full Event* In case of a lossless buffering strategy, an Ethernet frame is transmitted once the queue is full (Scharbarg et al. 2005; Nacer et al. 2013; AUTOSAR 2015).
- *Trigger Frames* Incoming CAN frames with particular identifiers immediately trigger the transmission of an Ethernet frame, which corresponds to the AUTOSAR configuration TriggerMode=TRIGGER_ALWAYS (Kern et al. 2011; AUTOSAR 2015).
- *Per-frame Timeout* Incoming CAN frames set a timer, which then triggers the transmission of an Ethernet frame, i.e. to bound the sampling delay for these frames. This resembles the timed approach in Kern et al. (2011).

The last three trigger mechanisms add dependencies between all CAN frames within a buffer, i.e. the trigger condition depends on the arrival pattern of incoming CAN frames. These dependencies can be reduced by introducing multiple buffers to partition the CAN traffic (mapping), according to e.g. destination, priority, or deadline. Due to their complexity, we do not model per-frame timeouts (cf. Sect. 5.3.6 for a discussion).

CAN frames can be mapped to a buffer statically or dynamically. A static mapping assigns a CAN identifier to a buffer at design time so that each CAN frame with this identifier is always stored in the same buffer. In a dynamic mapping the assignment is not predefined and is instead determined at run-time. In such an approach the CAN frames would not be mapped statically to a fixed buffer (or set of buffers) but the gateway would rather decide at run-time which output buffer will be most suitable for an incoming CAN frame. The eligible output buffers are constrained by the CAN frame's destination(s). For this, a set of reasonable (online) strategies should be developed. However, before a formal analysis can be applied, it must be evaluated if reasonable bounds for the worst-case behavior of these online strategies can be derived. In this article, we focus on static mapping only, as dynamic mapping introduces an additional level of indeterminism and complexity to a formal analysis.

5.3 Multiplexing event models

We model multiplexing as an event model operator which depends on the semantics of the join (i.e. its triggering mechanism) and its input event models. More formally, we define the multiplexing function $\mathcal{M} : \boldsymbol{\Delta} \rightarrow \hat{\boldsymbol{\delta}}$ that, given a set of input event models $\boldsymbol{\Delta} = \{\boldsymbol{\delta}_0, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_n\}$, calculates the combined output event model $\hat{\boldsymbol{\delta}}$ (cf. Fig. 4). In this section, we consider buffer timeouts, trigger frames, buffer-full events, and combinations of these.

5.3.1 Buffer timeout

This triggering mechanism is modeled by a periodic event model $\boldsymbol{\delta}_{to}$ whose period corresponds to the timeout value t_{to}. The output event model $\hat{\boldsymbol{\delta}}$ is solely determined by the timeout event model.

$$\hat{\boldsymbol{\delta}} = \boldsymbol{\delta}_{to} \quad \text{with} \quad \begin{aligned} \delta^-_{to}(n) &= (n-1) \cdot t_{to} \\ \delta^+_{to}(n) &= (n-1) \cdot t_{to} \end{aligned} \tag{2}$$

5.3.2 Trigger frames

Let $\boldsymbol{\Delta}_{tr} \subseteq \boldsymbol{\Delta}$ be the subset of triggering event models. Then, the output event model is determined by event models $\boldsymbol{\delta}_i \in \boldsymbol{\Delta}_{tr}$. As a trigger frame causes the immediate emission of an output event, an Ethernet frame can only contain, among other non-trigger frames, a single trigger frame. The output event model $\hat{\boldsymbol{\delta}}$ is modeled by an OR-join (Henia et al. 2005) of all triggering event models. The OR event model is conveniently modeled in the η domain, which can be converted to the δ domain to retrieve $\hat{\boldsymbol{\delta}}$ (Diemer et al. 2012a).

$$\begin{aligned} \hat{\eta}^+(\Delta t) &= \sum_{\eta_i \in \boldsymbol{\Delta}_{tr}} \eta^+_i(\Delta t) \\ \hat{\eta}^-(\Delta t) &= \sum_{\eta_i \in \boldsymbol{\Delta}_{tr}} \eta^-_i(\Delta t) \end{aligned} \tag{3}$$

5.3.3 Trigger frames with buffer timeout

Here, we consider triggered frames in conjunction with a buffer timeout. If no trigger frame arrives, the data is sent after t_{to} at the latest. We interpret the buffer timeout as a triggering event model, and add it to the set of triggering event models $\boldsymbol{\Delta}_T = \boldsymbol{\Delta}_{tr} \cup \{\boldsymbol{\delta}_{to}\}$. The resulting event model can be derived analogously to Eq. 3.

This approach may introduce a certain degree of over approximation. A practical implementation would certainly omit an Ethernet frame transmission on a buffer timeout event if the buffer is empty. There are, however, also scenarios in which this behavior is desired, e.g. in lossy buffering when the latest known values should be distributed periodically (in this case the buffer is never empty but always keeps the latest values). Also, one may decide to optimize the implementation by resetting the timer once a CAN frame is placed into an empty buffer as specified by AUTOSAR (2015).

5.3.4 Buffer-full

At this point, we are able to analyze multiplexing scenarios with lossy buffering. For lossless buffering, however, we also need to take buffer-full events into account.

Let $\boldsymbol{\Delta}_B \subseteq \boldsymbol{\Delta}$ be the subset of all event models which are non-triggering and queued in the buffer. Let $\boldsymbol{\delta}_B$ denote the OR-join (cf. Eq. 3) of all input event models in $\boldsymbol{\Delta}_B$, and let m denote the size of the buffer in number of CAN frames. Now, we can construct best-case and worst-case scenarios to derive the combined output event model $\hat{\boldsymbol{\delta}}$. Suppose that at time t_0 the buffer has just been filled with $\tilde{m} \in [0, m]$ CAN frames (i.e. frame $\tilde{m}$ just arrived), such that $m - \tilde{m}$ additional frames are required to transmit the buffer. This implies that the last buffer transmission was $\tilde{m} + 1$ frames before t_0 (one frame to transmit the buffer and then $\tilde{m}$ frames to fill the buffer up to time t_0). So, two output frames (of $\hat{\boldsymbol{\delta}}$) are $\tilde{m} + 1 + m - \tilde{m} = m + 1$ input frames (of $\boldsymbol{\delta}_B$) apart. In the worst-case/best-case the minimum/maximum-distance between these two input frames is described by $\delta_B^-(m+1)/\delta_B^+(m+1)$. This argumentation can be generalized to:

$$n \geq 2: \quad \begin{aligned} \hat{\delta}^-(n) &= \delta_B^-((n-1)m+1) \\ \hat{\delta}^+(n) &= \delta_B^+((n-1)m+1) \end{aligned} \tag{4}$$

5.3.5 Buffer-full, trigger frames, buffer timeout

Here, we combine the buffer-full event with trigger frames and the buffer timeout. Let $\boldsymbol{\delta}_{tr}$ denote the OR-join of $\boldsymbol{\Delta}_T$ which contains all triggering event models, i.e. trigger frames and buffer timeout (cf. Eq. 3). Any n frames observed at the output of the multiplexer can be decomposed into a number of frames triggered by single trigger events n_{tr} and a number of frames triggered by buffer-full events n_{buf}, i.e. $n = n_{tr} + n_{buf}$. Now, the timing of the output events depends on the number of trigger frames and buffer-full events:

$$\begin{aligned} \hat{\delta}^-(n_{buf}, n_{tr}) &= \max\{\delta_B^-((n_{buf}-1)\cdot m+1), \delta_{tr}^-(n_{tr})\} \\ \hat{\delta}^+(n_{buf}, n_{tr}) &= \max\{\delta_B^+((n_{buf}-1)\cdot m+1), \delta_{tr}^+(n_{tr})\} \end{aligned} \tag{5}$$

As the decomposition $n = n_{tr} + n_{buf}$ is unconstrained and any combination can possibly occur, we must find the worst-case/best-case combination:

$$\begin{aligned} \hat{\delta}^-(n) &= \min\{\hat{\delta}^-(n_{buf}, n_{tr}) \mid n_{buf} + n_{tr} = n\} \\ \hat{\delta}^+(n) &= \min\{\hat{\delta}^+(n_{buf}, n_{tr}) \mid n_{buf} + n_{tr} = n\} \end{aligned} \tag{6}$$

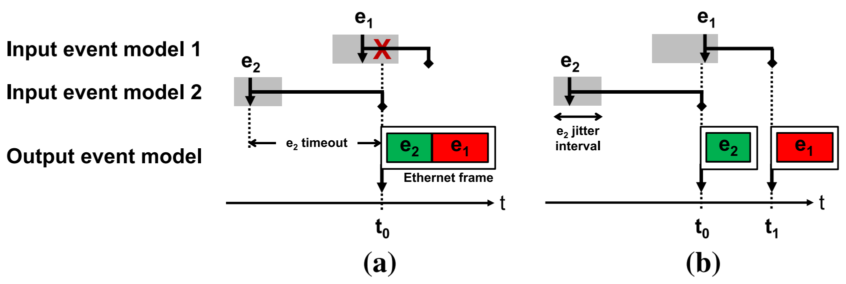

Fig. 5 Per-frame timeout examples

Note that $\hat{\delta}^+(n)$, like its counterpart, is the minimum over all combinations. This is because no trigger condition that occurs within the considered time window must be missed.

5.3.6 Per-frame timeouts

In the per-frame timeout triggering strategy, CAN frames can be associated with an individual timeout, which then triggers the transmission of an Ethernet frame. When such a CAN frame arrives at a gateway, it is guaranteed to be sent out (as part of an Ethernet frame) after at least its timeout, i.e. the timeout allows to specify an upper bound on the CAN frame's residence time in the gateway.

Per-frame timeout triggering can be approximated by the buffer timeout strategy by using the minimum of all frame timeouts as the buffer's timeout δ_{to}. This approach assumes that, in the worst-case, a CAN frame of the stream with the shortest timeout arrives just after the buffer has been sent (cf. e_1 Fig. 5b). Care must be taken when deriving the sampling delay for each steam (cf. Sect. 5.4 below). By definition of per-frame timeouts, the worst-case sampling delay is the frame's timeout. However, this approximation is clearly an overestimation.

Deriving tighter worst-case output event models for per-frame timeouts is hard, due to the complex interaction of the input event streams in this strategy. The triggering strategies we presented so far have in common that the trigger condition, which releases an Ethernet frame, occurs concurrently with the arrival of an input event, e.g. a timeout which acts as the exclusive trigger or the event which triggers the buffer to be sent in the buffer-full strategy. In a per-frame timeout strategy, however, this trigger condition occurs some time (i.e. the timeout) after the corresponding input event arrives. Depending on the arrival pattern of the input events and their timeouts, there can be masking effects, where one frame completely masks the actions of another frame. An example is shown in Fig. 5 where the arrival time of event e_1 (within its jitter interval) determines whether it is masked by event e_2's timeout, i.e. it is part of the Ethernet frame which is triggered by e_2's timeout (at time t_0 in Fig. 5a), or contributes to the output event model by triggering another Ethernet frame (at t_1 in Fig. 5b).

Hence, to derive the worst-case output event model, we must consider the relative arrival times of the events of all input event models. Per input event, these times can be anywhere within its jitter interval. Additionally, the relative position of the jitter intervals of different input event models can be affected by an offset between these input event models. This problem is complex to solve, as we have to consider all event arrival times within their respective jitter intervals together with all offsets between the input event models to find the worst-case output event model.

If we introduce certain assumptions to simplify this problem, we can outline a constructive method to derive output event models for per-frame timeouts. First, assuming strictly periodic input event models can be used to eliminate the jitter interval problem. For each input event model, this assumption implies that the timeouts within this input event model are also periodic with the input event model's period, i.e. the (minimum) distance between any two timeouts of this input event model also is the input event model's period. Second, only trying to find the occurrence of an input event's next timeout without considering its history beyond its last timeout can be used to eliminate the offset problem between different input event models. Now, given an output event, for each per-frame timeout event model, the occurrence of its next timeout can be derived by taking into account that this timeout must occur (a) at least one period after the input event's last timeout or (b) by its timeout after the last output event (as in Fig. 5b), whichever occurs last. The next output event occurs at the minimum over all these new timeouts.

Event models from real CAN buses typically have jitter, such that this method cannot be applied directly. Moreover, if we relaxed the first assumption and allowed jitter, then the minimum distances between the timeouts of the per-frame timeout event models would decrease. If the jitter grows too large, in our argumentation above, case (b) will dominate case (a), which is equivalent to the initially presented overestimation of using the minimum of all frame timeouts as the buffer's timeout.

5.4 Sampling delay

As discussed in Sect. 3, the end-to-end latency along a path can be computed by summing up the response times of the contained tasks (cf. Eq. 1). The multiplexing join, however, induces an additional delay for non-trigger frames because an input event does not trigger an output event immediately. This is similar to a scenario where the inputs of a subsystem are periodically sampled, which results in a time-triggered rather than an event-triggered activation of the subsystem (Feiertag et al. 2008). In the multiplexing scenario, we therefore add a similar sampling delay (denoted L_i^s) to the end-to-end latency of non-trigger frames, which results from the maximum distance between two output events of the corresponding multiplexing join:

$$L_i^s = \begin{cases} 0 & \text{if } \boldsymbol{\delta}_i \in \boldsymbol{\Delta}_{tr} \\ \hat{\delta}^+(2) & \text{otherwise} \end{cases} \tag{7}$$

5.5 Demultiplexing event models

The demultiplexing problem is the inverse of the multiplexing problem: Given a combined input event model $\hat{\boldsymbol{\delta}}'$, we have to find the set of individual output event models

$\mathbf{\Delta}' = \{\boldsymbol{\delta}'_0, \boldsymbol{\delta}'_1, \ldots, \boldsymbol{\delta}'_n\}$ (cf. Fig. 4). The individual output event models $\boldsymbol{\delta}'_i$, in turn, are defined by their lower and upper bounds. In the following, we discuss how these bounds can be derived conservatively. We start with a naive approach to illustrate the general demultiplexing problem before we proceed to present a more sophisticated approach that is guaranteed to preserve the load on the egress CAN bus.

5.5.1 Naive bounds

The *naive* approach is based on the idea that each combined event at the demultiplexer (incoming Ethernet frame) can induce multiple demultiplexed events (CAN frames) for each stream. Hence we can establish the *naive* bounds by scaling the combined event model with the minimum/maximum number of contained events for the particular stream.

For lossy buffering, there can be at most a single CAN frame of each stream, as, during multiplexing, old CAN frames will always be updated (overwritten) by the most recent ones. Hence, the upper bounds of the demultiplexed event models $\boldsymbol{\eta}'_i$ on the egress side follow the upper bound of the combined input event model from the incoming Ethernet frame: $\eta_i'^{+}(\Delta t) = \hat{\eta}'^{+}(\Delta t)$. The number of output events of a particular outgoing event model interface is lower bounded by zero at all times, i.e. $\eta_i'^{-}(\Delta t) = 0$. Again, $\boldsymbol{\eta}'_i$ can be converted to $\boldsymbol{\delta}'_i$ (Diemer et al. 2012a).

Under lossless buffering, the number of CAN frames in a buffer (incoming Ethernet frame) depends on the multiplexing strategy ($\mathcal{M}$) and the arrival pattern of the CAN frames ($\boldsymbol{\delta}_i$) on the ingress gateway. We can derive an upper bound for the number of CAN frames of each stream j by assuming that the CAN frames of this stream arrived as fast as possible (δ_j^-), while the CAN frames of all other streams $i \neq j$ arrived as late as possible (δ_i^+). This represents the worst possible combination of event arrivals w.r.t. the number of events from stream j, as events from stream j cannot arrive faster and events from other streams cannot arrive slower. More generally, we let $\mathbf{\Delta}_j$ denote this set of event models so that we can calculate a combined output event model $\hat{\boldsymbol{\delta}}_j$ based on these assumptions dependent on the multiplexing strategy: $\hat{\boldsymbol{\delta}}_j = \mathcal{M}(\mathbf{\Delta}_j)$. We can then bound the number of CAN frames N_j of stream j from above by calculating the number of events of stream j that can arrive between two output events: $N_j = \eta_j^+(\hat{\delta}_j^+(2))$. Thus, at the egress gateway, the maximum number of CAN frames of stream j in a single Ethernet frame can be upper bounded by scaling the combined input event model by N_j: $\eta_j'^{+}(\Delta t) = N_j \hat{\eta}_j'^{+}(\Delta t)$. As in the lossy case, the number of output events of a particular outgoing lossless event model interface is lower bounded by zero at all times, i.e. $\eta_j'^{-}(\Delta t) = 0$.

For both cases, lossy and lossless, this is an overly pessimistic approximation, as it (conservatively) underestimates/overestimates the minimum/maximum number of CAN frames for each multiplexed stream inside every incoming Ethernet frame without respecting any correlations between the scenarios. E.g. assuming the maximum number of CAN frames for each multiplexed stream (per incoming Ethernet frame) can lead to mutually exclusive worst-case scenarios for the upper event bounds: under buffer-full buffering, for example, we generally have $\sum_{0 \leq i < |\Delta|} N_i \geq m$, where m is the size of the buffer (number of CAN frames per Ethernet frame). Consequently, by using this naive approach, the (artificially) increased number of CAN frames on the egress side potentially induces a higher load and jitter on the egress gateway and CAN bus(es). This is a major drawback as we have observed that (especially) CAN buses can be easily overloaded, i.e. deemed unscheduable, by using this approach. Although this could be improved by deriving bounds on the number of CAN frames within multiple consecutive Ethernet frames instead of applying the same N_j on every single Ethernet frame, we present another approach that is guaranteed to preserve the load on the egress gateway and CAN bus(es) in the next section.

5.5.2 Load-preserving bounds

This approach is based on the assumption that the number of events entering the join must be equal to the number of events exiting the fork, i.e. we reasonably assume that no events are lost or added during multiplexing, transmission, and demultiplexing. It is similar to Perathoner et al. (2010) and Rox and Ernst (2008) as it considers a hierarchical structure of event models, i.e. inner event models that are propagated along with their outer event model. More precisely, we propagate the original (inner) input event models $\boldsymbol{\delta}_i \in \boldsymbol{\Delta}$ of the join along with its (outer) output event model $\hat{\boldsymbol{\delta}}$ and add the path jitter to the inner event models upon their extraction at the demultiplexing fork. Note, that under lossy buffering this approach can introduce a slight overestimation in rare cases where there might actually be desired frame loss, e.g. when the buffer timeout at the join is configured to exceed a stream's period.

Let J_i^{JF} be the path jitter, which is defined as the difference between the worst-case and best-case end-to-end latency (incl. the sampling delay L_i^s) of a single event of path i along the subpath $\hat{i}$ from the join to the fork (cf. Fig. 4 and Eq. 1):

$$J_i^{JF} = l_i^+ - l_i^- \quad \text{with} \quad \begin{aligned} l_i^+ &= L_i^s + L_{\hat{i}}^+(1) \\ l_i^- &= 0 + L_{\hat{i}}^-(1) \end{aligned} \tag{8}$$

The path jitter is then applied as described by Rox and Ernst (2008):

$$\begin{aligned} \delta_i'^-(n) &= \max(\delta_i^-(n) - J_i^{JF}, 0) \\ \delta_i'^+(n) &= \delta_i^+(n) + J_i^{JF} \end{aligned} \tag{9}$$

6 Experiments

Here, we evaluate the presented gateway analysis in a more complex system topology that combines video, control, and inter-gateway traffic. Our comparison metrics are the worst-case end-to-end latency guarantee and load from our analysis.

6.1 System topology

Figure 6 depicts the synthetic system topology on which we based our experiments. It comprises an Ethernet network of one inner switch (S center) and four outer switches

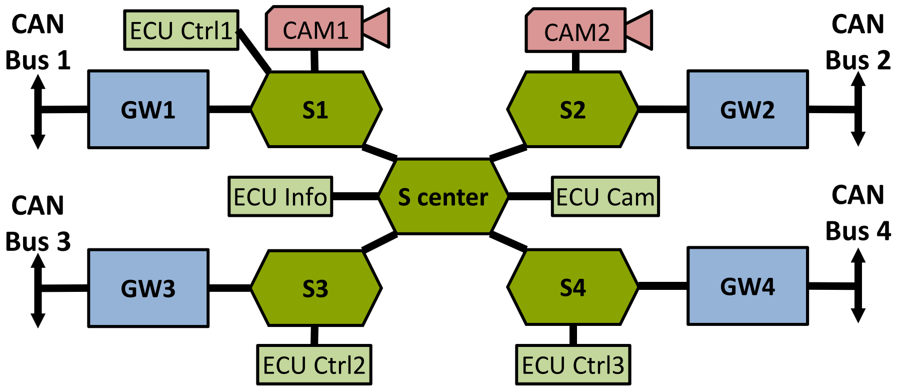

Fig. 6 System topology for our experiments comprising a network with five switches, four gateways, two cameras and five ECUs. Ethernet links are bidirectional 100 Mbit/s links.eps

Table 1 Ethernet traffic: three video streams and two control streams

Name	Source	Destination	PCP	Payload (bytes)	Period	Jitter
Video 1	CAM1	ECU Cam	2	786	250 µs	0
Video 2	CAM2	ECU Cam	2	786	250 µs	0
Video 3	ECU Cam	ECU Info	2	786	250 µs	0
Control 1	ECU Ctrl1	ECU Ctrl3	3	50	10 ms	10 ms
Control 2	ECU Ctrl2	ECU Ctrl3	3	100	50 ms	50 ms

PCP denotes the priority (IEEE 802.1Q)

(S1–S4). Moreover, there is a CAN/Ethernet gateway connected to each of the outer switches. Each of these gateways, in turn, connects a CAN bus to the system. Two cameras are assigned to S1 and S2. Further, S1, S3, and S4 each connect to a dedicated ECU. There are also two additional ECUs which are directly linked to the inner switch. All these components are interconnected by bidirectional 100 MBit/s links.

6.2 Traffic streams

Table 1 specifies the *Ethernet traffic* streams in the network. There are two video streams originating from the cameras that are received and processed by ECU Cam. This ECU sends the resulting video stream to the Infotainment ECU (ECU Info). Two additional control traffic streams are sent from ECU Ctrl1 and ECU Ctrl2 to ECU Ctrl3. Control traffic is mapped to a high Ethernet priority (PCP), while video traffic is mapped to a lower one.

To model the *inter-gateway CAN traffic* streams between the gateways, we use a synthesized set of 100 periodic CAN frames with periods between 16 and 1001 ms. The periods and CAN identifiers of these frames are assigned by random according to distributions that we observed in real automotive systems. We divided this set further into two disjunct sets, which are sent from GW1 to GW3 and from GW2 to

Table 2 Multiplexed inter-gateway traffic by the source and destination gateway, the buffer size and the buffer timeout

#	Source	Destination	Buffer size	Timeout (ms)	Period range (ms)
1	GW1	GW3	10	20	0–30
2	GW1	GW3	10	100	90–110
3	GW1	GW3	10	120	110–130
4	GW1	GW3	10	1000	900–1010
5	GW2	GW4	10	40	30–50
6	GW2	GW4	10	60	50–70
7	GW2	GW4	10	80	70–90
8	GW2	GW4	10	140	130–150
9	GW2	GW4	10	280	270–290
10	GW2	GW4	10	480	470–490

A period range is specified, to group similar CAN frames

Table 3 Number of trigger and total number of CAN frames assigned to the multiplexing groups

Group #	1	2	3	4	5	6	7	8	9	10	Total
# Assigned frames	17	12	17	5	6	6	10	14	9	4	100
# Trigger frames	0	0	1	1	0	0	1	0	1	0	4

WG4, respectively. The former set contains 51 CAN traffic streams, which induce an asymptotic bus load, i.e. the (theoretical) average load over a time interval of infinite length, of about 31 % on a 500 kBit/s CAN bus, while the latter set comprises 49 streams, which induce an asymptotic bus load of about 14 %. For the later evaluation, we set the jitter of the CAN frames to half their periods to accommodate for delays within parts of the system that we do not model.

We manually assembled these inter-gateway CAN traffic streams into ten multiplexing groups as shown in Table 2. For every stream we assume an UDP/IP transport protocol (i.e. 28 bytes of additional protocol overhead) and assign it Ethernet priority 3, i.e. the same priority, we used for control traffic in Table 1. The CAN frames of these streams are statically mapped to the Ethernet frames of Table 2 according to their period, which must match the specified range. The period ranges have been chosen to match (rounded down/up to the nearest 10 μs) the minimum/maximum periods of each group's traffic streams. For a given mapping, the Ethernet frame size can be computed as described by Diemer et al. (2012b). We also selected some of the CAN frames to be trigger frames. Table 3 shows how many frames have been assigned to which group based on this mapping scheme. Note that all frames from GW1/GW2 are transmitted to GW3/GW4 respectively, which later allows a direct comparison of the ingress and egress sides.

For each group, we instantiate a multiplexer on the source and a demultiplexer on the destination gateway (cf. Fig. 4). Assuming a multiplexing strategy using buffer timeouts, trigger frames, and buffer-full events, the output event models can be derived

Table 4 Worst-case latency for video and control streams. 1:1 mapping, multiplexing (mux) and all-in-one mapping

Stream:	Control 1 (μs)	Control 2 (μs)	Video 1 (μs)	Video 2 (μs)	Video 3 (μs)
1:1	825	422	671	638	68
mux	349	201	333	313	68
All-in-one	246	131	299	280	68

Table 5 Shared network resources (switch ports)

Port	Traffic streams
S1→S center	Video 1, Control 1, GW1→GW3
S2→S center	Video 2, GW2→GW4
S center→ECU Cam	Video 1, Video 2
S center→S4	Control 1, Control 2, GW2→GW4
S4→ECU Ctrl3	Control 1, Control 2

from the arrival patterns of their associated CAN frames. The best-case/worst-case execution times of the ingress gateway tasks (cf. Sect. 5.1) are estimated to be 10 μs/20 μs for the RX tasks and 10 μs/50 μs for the TX tasks. Similarly, the execution times of the egress gateway tasks are assumed to be 10 μs/20 μs for the TX tasks and 10 μs/50 μs plus $N_j \cdot 5$ μs for the RX tasks. For the latter, we additionally account for processing overhead depending on the maximum number N_j of packaged CAN frames within a single Ethernet frame of group j (cf. Sect. 5.5.1).

6.3 Results

We evaluate our analysis by comparing the multiplexing scenario defined by Tables 2 and 3 (i.e. buffer timeouts, trigger frames, and buffer-full events) to the two extrema discussed in the introduction: a 1:1 mapping (i.e. transmitting each CAN frame separately) as well as to an all-in-one mapping (i.e. transmitting as many CAN frames together as possible). More precisely, the latter scenario is defined by two multiplexing groups, one that transmits all CAN frames from GW1 to GW3 and one for the CAN frames from GW2 to GW4. There is no buffer timeout in the all-in-one scenario whereas the buffer size is raised to 30. In all scenarios, we use the load-preserving demultiplexing approach from Sect. 5.5.2.

Table 4 shows the resulting end-to-end latency guarantees of the Ethernet traffic streams listed in Table 1 for all three scenarios. As the inter-gateway traffic is assigned the highest Ethernet priority, it interferes with the video and control streams. The points of possible interference (i.e. shared resources) are listed in Table 5. As multiplexing reduces the number of traffic streams injected by the gateways considerably, the interference with other traffic streams decreases as well. Hence their end-to-end latency improves by at least 50 % except for stream Video 3, which does not experience any

Table 6 Load comparison between 1:1 mapping, multiplexing (mux) and all-in-one mapping on the gateways and the ports of the central switch

Res.:	GW1 (%)	GW2 (%)	GW3 (%)	GW4 (%)	S center→ ECU Cam (%)	S center→ ECU Info (%)	S center →S3 (%)	S center →S4 (%)
1:1	8.2	3.7	8.2	3.7	54.9	27.4	0.78	0.48
mux	3.3	1.7	4.2	2.3	54.9	27.4	0.23	0.28
All-in-one	2.6	1.2	3.3	1.7	54.9	27.4	0.12	0.21

interference from other streams. As expected, the all-in-one scenario minimizes the interference on other streams.

Table 6 shows the asymptotic load for the gateways and selected ports on the central switch as a result of the worst-case analysis. Due to the low bandwidth requirements of the inter-gateway traffic, there is only a small improvement on the network load in the multiplexing scenario. The gateway load, in contrast, is significantly lower under multiplexing, as less Ethernet packets must be processed. Note that the load increase on the egress gateways (GW3 and GW4) results from the higher execution times of the RX tasks (which depend on the number of CAN frames in an Ethernet frame) in comparison to the corresponding TX tasks on the ingress gateways (GW1 and GW2). This additional load, however, is only required during the demultiplexing process on the egress gateway. The asymptotic load induced by the demultiplexed CAN streams on the egress CAN buses (CAN buses 2 and 4), in contrast, is preserved and equals the asymptotic load on the corresponding ingress CAN buses.

Each multiplexing strategy introduces a certain amount of jitter, which is caused by the sampling delay (if present) and the Ethernet latency difference during frame transmissions. This jitter can (potentially) lead to transient overload peaks on the egress side. Figure 7 illustrates how the different approaches affect the (transient) worst-case load induced by the inter-gateway CAN frames on the pretended ingress and egress CAN buses. Specifically, the figure shows the transient worst-case load (on the y-axis) versus the time interval in which this load can be observed (on the x-axis). Note that, as the traffic on ingress CAN buses 1 and 2 is not multiplexed, all approaches perform the same on these two buses. As expected, the latency a CAN frame experiences on the path between the ingress and egress gateway results in additional output jitter and therefore (potentially) a bursty arrival at the egress CAN bus. Asymptotically, there is no difference between the load on the ingress and egress buses as our analysis is load preserving, i.e., although not shown, the asymptotic load on egress buses 3 and 4 approaches loads of 31 and 14 % (respectively) as the time interval approaches infinity. The lowest (transient) overload can be seen for the 1:1 mapping whereas the all-in-one mapping results in the highest overload. This is also reflected in the latency results for the inter-gateway traffic.

Figure 8 shows the worst-case end-to-end (i.e. gateway-to-gateway) latency guarantees of the inter-gateway traffic in all scenarios *relative to the periods* of the frames. The frames are sorted by their period. In 1:1 mapping, the worst-case response times of the corresponding gateway tasks dominate, which, in turn, are affected by their priority

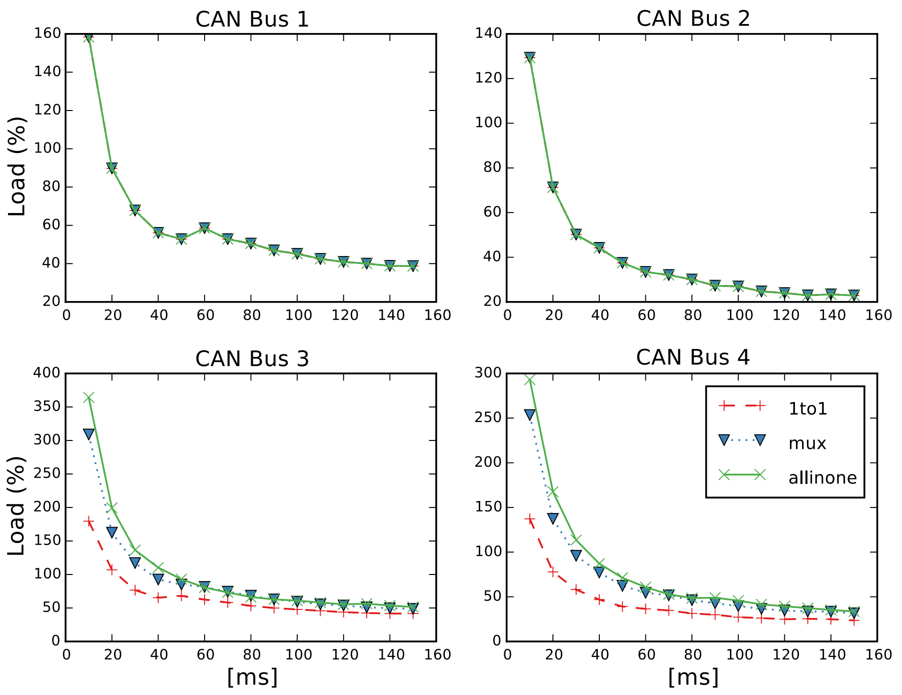

Fig. 7 Worst-case transient load on the (pretended) ingress (Bus 1, Bus 2) and egress (Bus 3, Bus 4) CAN buses

assignment (CAN frames with lower IDs have higher priority). This becomes clear by looking at the frames of the first group (first 17 bars in Fig. 8) that all have similar periods of 16–23 ms but quite different latencies (cf. Fig. 9, which shows the absolute latencies for the 8-th (#671) and 9-th (#1854) frames of the group). Looking at the longer-periodic frames, we can identify some room for improvement to exploit in the multiplexing scenario as their latency is much lower than their period. In the multiplexing scenario, for non-trigger frames, the buffer timeouts, i.e. the sampling delays, dominate. The latency of trigger frames (#1420, #822, #1966, #1462) is improved over the latency from 1:1 multiplexing (e.g. compare the absolute latencies of frame #1420 in Fig. 9a, b). Contrastingly, the all-in-one scenario is dominated by the buffer-full events and trigger frames, resulting in large latencies for the lower-periodic frames. Figure 8 clearly shows that we can influence the frame latencies (in both directions) by multiplexing. We will continue this train of thought when we discuss the impact of different multiplexing parameters in Fig. 10.

Figure 9 depicts the latency contributions of the Ethernet, the ingress and egress gateways, as well as the sampling delay for four selected frames and thereby further illustrates the dominating factors. The frames are selected from two different groups in the multiplexing and all-in-one scenario. We notice that the sampling delay in Fig. 9 equals the corresponding buffer timeouts and that the trigger frame #1420 is not affected by the sampling delay. In the all-in-one scenario (Fig. 9), we observe that frames #671 and #1854 experience a sampling delay of ∼53 ms that results from the

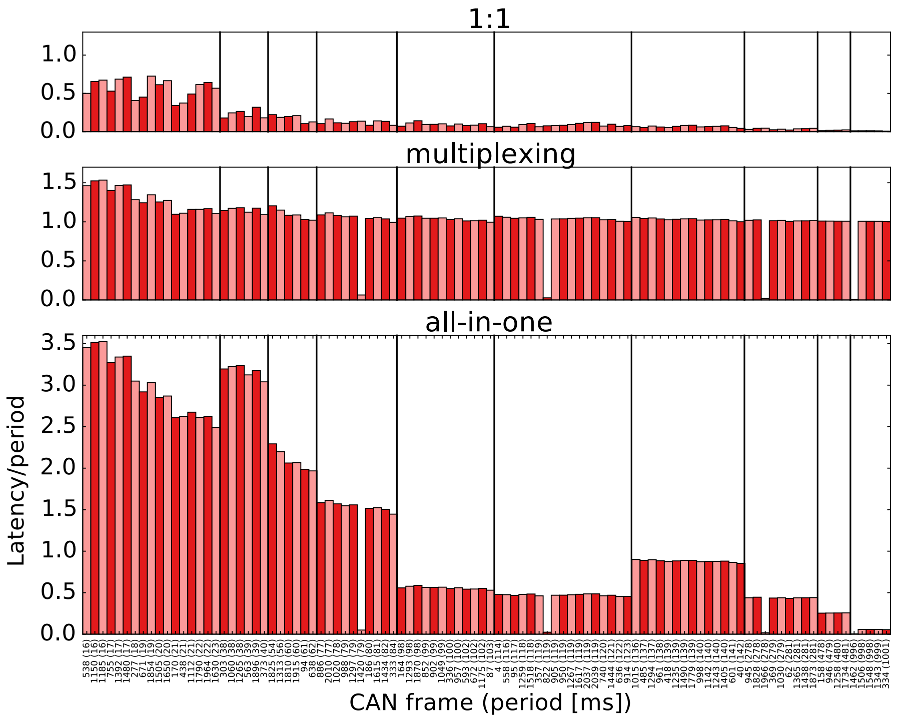

Fig. 8 Period-relative worst-case end-to-end latency of CAN frames in the 1:1, multiplexing, and all-in-one scenarios. The *vertical lines* represent the group boundaries from Table 2

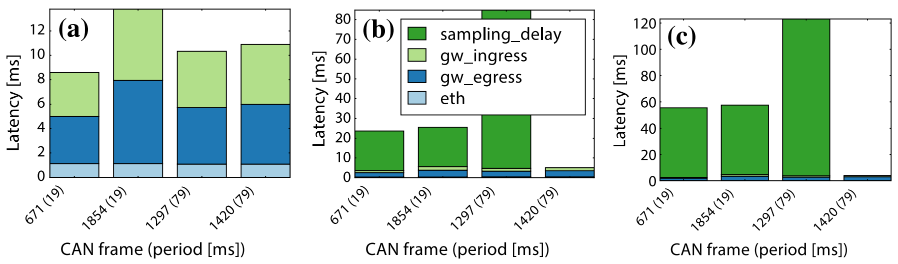

Fig. 9 Worst-case end-to-end latency contributions of selected CAN frames for 1:1 mapping (**a**), multiplexing (**b**), and all-in-one mapping (**c**)

buffer-full event whereas frame #1297 has a sampling delay of ∼119 ms resulting from the period (79 ms) and jitter (39.5 ms) of trigger frame #1420.

Next, we evaluate the effect of different multiplexing parameters on the end-to-end latencies guarantees. In the following, we focus on frame #822, which is the only trigger frame of group 3. The buffers for all groups in Table 3 are evaluated for sizes 2, 5, 10, 15, and 20 (named BUF2 to BUF20). We vary the timeouts for each group within 0, 25, 50, 75, and 100 % of their period range (named TO0.0 to TO1.0). For group 3, 0 % corresponds to 110 ms and 100 % to 130 ms. At the same time, these

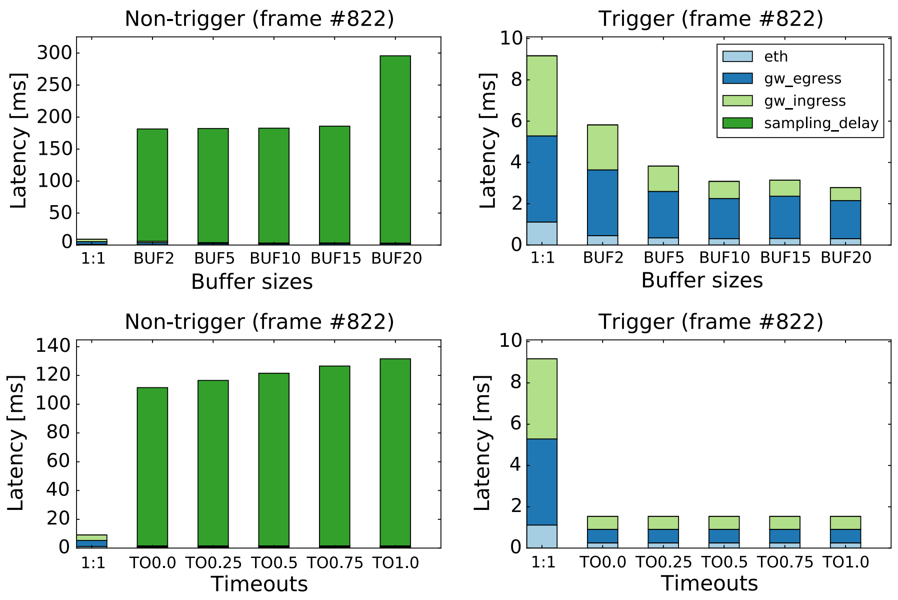

Fig. 10 Worst-case end-to-end latency guarantees of frame #822 for different buffer sizes and timeouts

parameters lead to 90 and 110 ms for group 2. For group 1 the 0 % bound is set to 5 ms to prevent overload at the gateways.

Figure 10 shows the effect of these parameters on the end-to-end latency guarantees of frame #822. The subgraphs on the right show the latencies when frame #822 is considered as a trigger frame (as originally specified), whereas for the subgraphs on the left, we consider frame #822 as a non-trigger frame, i.e. in this scenario group 3 does not contain any trigger frames.

The latencies of the non-trigger setup are dominated by the sampling delay $\delta^+(2)$, which is the longest distance between any two output events (Eq. 7). Thus, during timeout evaluation, the sampling delay corresponds to the timeouts. In the buffer size evaluation, the sampling delay increases with buffer size, but there is a plateau between BUF2 and BUF15. This is because many frames in group 3 have roughly the same $\delta^+(n)$ leading to a burst in $\delta^+_B(n)$ (Eq. 4), which is large enough to generate the output events for BUF2 to BUF15 at nearly the same time.

In the trigger setup, smaller buffers induce more gateway and Ethernet load, leading to higher interference, and thus, longer latencies. At the same time, as long as the buffer-full events are dominating, a larger buffer size also increases the execution time of the RX tasks on the egress gateway in our model, as this time depends on the number of CAN frames in an incoming Ethernet frame. At BUF15, we can observe that the increased execution time overbalances the reduced interference. From BUF15, the group's trigger event begins to dominate buffer-full events. There is effectively no influence of different timeouts on the latency for frame #822. This is expected, as, in this scenario, frame #822 is a trigger frame whose period is in the order of the smallest timeout (TO0.0).

In all scenarios, the gateway load is lower than in the 1:1 scenario ($\sim 8\,\%$). Short timeouts and small buffers induce a high load, while large timeouts and big buffers result in lower loads.

6.4 Discussion

The focus of our evaluation was to demonstrate how multiplexing affects the design metrics worst-case end-to-end latency and load. In general (and unsurprisingly), latency can be reduced at the cost of increased load and vice versa. However, our formal analysis allows to quantify this trade-off, which enables future optimizations to find the best multiplexing parameters for a given problem.

Multiplexing (compared to a 1:1 mapping of CAN frames to Ethernet frames) can reduce the number of transmitted Ethernet frames, which leads to less load on the gateways (less frames require processing) and less interfering frames in the Ethernet network (cf. Table 6). Consequently, less interfering Ethernet frames lead to smaller end-to-end latencies of other (non-inter-gateway) Ethernet traffic (cf. Table 4). For inter-gateway traffic, however, multiplexing can introduce sampling delay, which negatively affects its end-to-end latencies (compared to a 1:1 mapping). This sampling delay can be controlled via different multiplexing parameters, e.g. timeouts or buffer sizes (cf. Figs. 8, 10). In general, smaller timeouts or smaller buffer sizes decrease the sampling delay at the cost of increased load and an increased number of interfering Ethernet frames. By making frames triggering, the sampling delay can be avoided entirely (cf. Figs. 8, 9). In fact, the 1:1 mapping is a scenario where all frames are trigger frames.

We have also shown that there can be complex interdependencies between different multiplexing parameters (cf. Fig. 10). Larger buffer sizes, for example, can reduce the number of Ethernet frames, which need to be processed. This can reduce the load, but at the same time more time is required at the egress gateway to extract individual CAN frames from larger Ethernet frames. Some triggering strategies also introduce dependencies of the output event models on the input event models, e.g. buffer-full triggering (cf. sampling delay plateau in the upper left subfigure of Fig. 10), while for others the output event models are independent of the input event models, e.g. buffer timeouts, or are dominated by a single trigger frame (cf. Fig. 10, lower right subfigure).

7 Conclusion

In this article, we presented a formal system-level analysis approach of heterogeneous, in-vehicle Ethernet-based systems, focusing on CAN-to-Ethernet gateway strategies. We considered complex frame packing effects with various triggering strategies such as buffer timeout, buffer-full as well as trigger frames. We modeled and analyzed an Ethernet-based backbone under different gateway multiplexing strategies. However, the additional latency introduced by gateway effects such as sampling latency is significantly larger than Ethernet effects. We showed that end-to-end latency can be adjusted by trading it for Ethernet and gateway load via different multiplexing parameters and vice versa. Formal analysis is able to reflect such design decisions enabling Ethernet network optimization guided by formal timing guarantees. This will help to establish Ethernet-based backbone networks for safety critical systems.

References

AUTOSAR (2015) Release 4.2.2, Specification of socket adapter

Ayed H, Mifdaoui A, Fraboul C (2011) Gateway optimization for an heterogeneous avionics network afdx-can. In: IEEE real-time systems symposium (RTSS)

Bauer H, Scharbarg JL, Fraboul C (2010) Improving the worst-case delay analysis of an afdx network using an optimized trajectory approach. IEEE Trans Ind Inf 6:521–533

Davis R, Burns A, Bril R, Lukkien J (2007) Controller area network (can) schedulability analysis: refuted, revisited and revised. Real-Time Syst 35(3):239–272

Diemer J, Rox J, Negrean M, Stein S, Ernst R (2011) Real-time communication analysis for networks with two-stage arbitration. In: Proceedings of EMSOFT, pp 243–252

Diemer J, Axer P, Ernst R (2012a) Compositional performance analysis in python with pycpa. In: Proceedings of WATERS. http://retis.sssup.it/waters2012/WATERS-2012-Proceedings.pdf

Diemer J, Rox J, Ernst R (2012b) Modeling of ethernet avb networks for worst-case timing analysis. In: Proceedings of MATHMOD

Feiertag N, Richter K, Nordlander J, Jonsson J (2008) A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In: Work on compositional theory and technology for real-time embedded systems CRTS

Henia R, Hamann A, Jersak M, Racu R, Richter K, Ernst R (2005) System level performance analysis—-the SymTA/S approach. IEE Proc Comput Digit Tech 152(2):148–166

Herber C, Richter A, Wild T, Herkersdorf A (2015) Real-time capable can to avb ethernet gateway using frame aggregation and scheduling. In: Proceedings of design, automation and test in Europe (DATE)

Kern A, Reinhard D, Streichert T, Teich J (2011) Gateway strategies for embedding of automotive CAN-frames into ethernet-packets and vice versa. In: Architecture of computing systems, pp 259–270

Lehoczky J (1990) Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proceedings of 11th RTSS, pp 201–209. doi:10.1109/REAL.1990.128748

Nacer A, Jaffres-Runser K, Scharbarg JL, Fraboul C (2013) Strategies for the interconnection of can buses through an ethernet switch. In: IEEE international symposium on industrial embedded systems (SIES), pp 77–80

Perathoner S, Rein T, Thiele L, Lampka K, Rox J (2010) Modeling structured event streams in system level performance analysis. In: ACM SIGPLAN/SIGBED conference on languages, compilers and tools for embedded systems (LCTES). ACM, Sweden, pp 37–46

Pop P, Eles P, Peng Z (2005) Schedulability-driven frame packing for multicluster distributed embedded systems. ACM Trans Embed Comput Syst 4(1):112–140

Revsbech K, Schiøler H, Madsen TK, Nielsen JJ (2011) Worst-case traversal time modelling of ethernet based in-car networks using real time calculus. In: Proceedings of NEW2AN, pp 219–230

Richter K (2005) Compositional scheduling analysis using standard event models. Ph.D. Thesis, TU Braunschweig

Rox J, Ernst R (2008) Modeling event stream hierarchies with hierarchical event models. In: Proceedings of design, automation and test in Europe DATE'08, pp 492–497. doi:10.1109/DATE.2008.4484729

Rox J, Ernst R (2010) Formal timing analysis of full duplex switched based ethernet network architectures. In: SAE World Congress, vol System level architecture design tools and methods

Saket R, Navet N (2006) Frame packing algorithms for automotive applications. J Embed Comput 2:93–102

Scharbarg J, Boyer M, Fraboul C (2005) Can-ethernet architectures for real-time applications. In: IEEE conference on emerging technologies and factory automation (ETFA), vol 2, pp 8–252

Schliecker S, Rox J, Ivers M, Ernst R (2008) Providing accurate event models for the analysis of heterogeneous multiprocessor systems. In: Proceedings of CODES-ISSS, pp 185–190

Thiele D, Axer P, Ernst R, Seyler JR (2014) Improving formal timing analysis of switched ethernet by exploiting traffic stream correlations. In: Proceedings of CODE+ISSS, New Delhi

Thiele L, Chakraborty S, Naedele M (2000) Real-time calculus for scheduling hard real-time systems. In: Proceedings of IEEE international symposium on circuits and systems (ISCAS), vol 4, pp 101–104. doi:10.1109/ISCAS.2000.858698

Zhang L, Gao H, Kaynak O (2013) Network-induced constraints in networked control systems—a survey. IEEE Trans Ind Inf 9(1):403–416