Formal Worst-Case Timing Analysis of Ethernet Topologies with Strict-Priority and AVB Switching

Jonas Diemer, Daniel Thiele, Rolf Ernst
Institute of Computer and Network Engineering
Technische Universität Braunschweig
38106 Braunschweig, Germany
{diemer|thiele|ernst}@ida.ing.tu-bs.de

***Abstract*—Ethernet is increasingly recognized as the future communication standard for distributed embedded systems in multiple domains such as industrial automation, automotive and avionics. A main motivation for this is cost and available data rate. A critical issue in the adoption of Ethernet in these domains is the timing of frame transfers, as many relevant applications require a guaranteed low-latency communication in order to meet real-time constraints. Ethernet AVB is an upcoming standard which addresses the timing issues by extending the existing strict-priority arbitration. Still, it needs to be evaluated whether these mechanism suffice for the targeted applications. For safety-critical applications, this can not only be done using intuition or simulation but requires a formal approach to assure the coverage of all worst-case corner cases. Hence, we present in this paper a formal worst-case analysis of the timing properties of Ethernet AVB and strict-priority Ethernet. This analysis mathematically determines safe upper bounds on the latency of frame transfers. Using this approach, we evaluate different topologies for a typical use-case in industrial automation.**

I. Introduction

Ethernet currently receives increasing interest for the use in embedded and distributed control applications in many different industries, such as industrial automation, automotive or avionics. There are multiple motivations for this. First of all, Ethernet delivers a much higher bandwidth than existing automation communication media such as CAN or FlexRay. At the same time, Ethernet hardware is extremely inexpensive in terms of per-node-cost due to the high quantities produced for the consumer market. Also, it offers straight-forward and efficient integration into other IT infrastructure.

One critical issue of the use of Ethernet for control applications, however, is that many such applications require predictable and low latency communication between nodes. In safety-critical applications, this has to be guaranteed even for the worst-case scenario. There are several approaches for Ethernet-based networks to make the timing more predictable which change the way data is forwarded in the bridges, e.g. by applying priorities. Examples are AFDX, Ethernet AVB, EtherCat, Profinet, TTEthernet [1]. Ethernet AVB has a special role in these approaches because it is an official Ethernet standard which is widely used. As a result, Ethernet AVB capable hardware is offered in large quantities and by different suppliers, which is beneficial in terms of costs and availability.

The Ethernet audio/video bridging (AVB) task group [2] has developed the Ethernet AVB standard over the past years to allow real-time audio and video streams for studio applications. The standardization work has resulted in the IEEE standards 802.1AS, 802.1Qat, 802.1Qav [3], [4], [5].

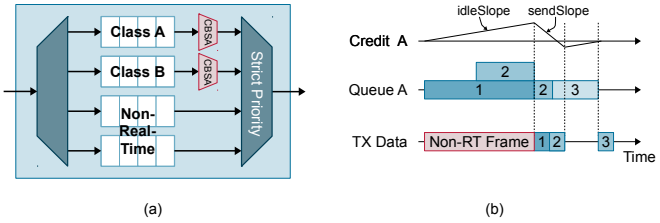

Fig. 1. Architecture (a) and operational example (b) of the Credit-Based Shaping Algorithm (CBSA) used in Ethernet AVB.

Compared to the previous Ethernet IEEE 802.1Q standard [6], which already provides strict priority scheduling with 8 priorities, Ethernet AVB adds a credit-based shaping algorithm (CBSA) for real-time (RT) traffic classes "A" and "B" as shown in Fig. 1(a). Each traffic class uses dedicated queues so scheduling within a class follows a FIFO order. The traffic shaper prevents starvation of lower priorities and allows bandwidth guarantees. For this it uses credits which are replenished at a constant rate (the so-called $idleSlope$) and consumed at the rate allowed by the port (the $sendSlope$) when data on the specific class is transferred. The $idleSlope$ is configured using the Stream Reservation Protocol (SRP) as defined by IEEE 802.1Qat.

A downside of Ethernet AVB is that the standard does not provide a formal latency guarantee, so this must be evaluated separately. The latency of communication largely depends on the network technology, topology and traffic scenarios. In order to evaluate the timing of specific Ethernet networks, simulation or measurements of real networks can be employed. These, however, are often rather slow, making design-space-exploration very cumbersome. More importantly, such approaches usually do not show the worst-case, and are thus not suitable to verify worst-case guarantees. Here, formal performance analysis approaches known from processor scheduling research seem like a better option. Compositional Performance Analysis (CPA) is one such approach which is very fast and highly scalable. In [7], we have shown how to apply CPA to

the analysis of Ethernet AVB using a model transformation. The key idea is to map the arbitration of frames at the output of Ethernet bridges to the scheduling of tasks on a processing resource. This way, we can utilize the well known methods for the performance analysis of distributed embedded systems provided by CPA to analyze the worst-case timing of Ethernet AVB.

While [7] focused mainly on the model transformation, this paper will comprehensively present the analysis of strict-priority Ethernet (IEEE 802.1Q) and Ethernet AVB class A and B in Section III. Before that, some further background and related work on AVB and CPA are provided in Section II. Furthermore, we will evaluate different topologies for their worst-case behavior comparing strict-priority Ethernet and Ethernet AVB for typical industrial use cases in Section IV before we conclude in Section V.

II. Background and Related Work

The right side of Fig. 1(b) illustrates the basic timing properties of Ethernet AVB by an example of the transfer of three short frames on Class A with a longer interfering non-real-time (NRT) frame. The top graph shows the credit level of the Class A traffic shaper. The middle graph shows the queue occupancy of Class A on the y-axis, while the boxes show the queuing delay of each of the three frames. The lower graph shows the data transmission on the output port. Frames on classes A or B are only sent if the corresponding credit level is zero or higher and the credit level is reset to zero as soon as there are no frames waiting on the corresponding queue. Hence, the traffic shaper enforces an idle time between consecutive frames in each class as shown between frames 2 and 3. When the corresponding traffic class is blocked due to a non-preemptive transfer that started earlier, credit can accumulate, resulting in a burst of frames once the output is free again as shown for frames 1 and 2.

The output arbitration in strict-priority Ethernet and AVB follows a static-priority non-preemptive (SPNP) scheduling scheme, for which local analyses exist (e.g. [8]). However, Ethernet allows different streams of the same priority to be scheduled in FIFO order, which requires an extension of the standard SPNP analysis. This extension is similar to the analysis of earliest-deadline-first (EDF), which is presented in e.g. [9]). For Ethernet without priorities, [10] presents a worst-case delay analysis. An analysis of Ethernet with strict priorities using Real-Time Calculus is given in [11]. This analysis, however, shows large overestimations. [12] presents a worst-case analysis of Ethernet with strict priorities and weighted fair queuing using network calculus. A study of the timing properties of Ethernet AVB has been presented by [13]. In this study, however, only per-class timings were obtained instead of those for individual streams.

In this paper, we use Compositional Performance Analysis (CPA) [14], [15], [16], which uses a similar composition and similar event models as Real-Time Calculus [17], but different local link and switch analyses. Real-Time Calculus has been used for Ethernet analysis in [11]. The CPA approach has been applied to regular Ethernet [18] and Ethernet AVB [7] by mapping Ethernet output ports to scheduled resources and traffic streams to chains of tasks. A similar approach has been shown for networks-on-chip in e.g. [19] and [20].

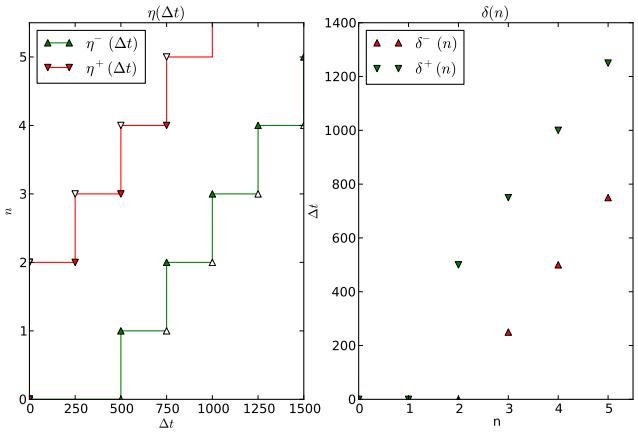

Fig. 2. Event model for a periodic task with a period of 250 time units.

CPA is implemented in the commercial tool SymTA/S, but the basic algorithms are also available open-source [21]. Compared to other formal approaches like model checking, CPA is very fast and scalable. The system model of CPA is composed of a set of tasks which are processed by a set of resources. Each task τ_i is assumed to consume processing time specified by its best- and worst-case execution times C_i^- and C_i^+ for each task activation (or job). Tasks are activated by events which can originate from an external source or from other tasks. Such events are modeled by minimum and maximum arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, which return the minimum and maximum number of events that can arrive within any time window of size Δt. These functions have pseudo-inverse counterparts, the so-called maximum and minimum distance functions $\delta^+(n)$ and $\delta^-(n)$, which are the maximum/minimum time interval between the first and the last event of any sequence of n event arrivals. An event model covers all possible event arrivals of a specific event source and is not just a specific trace of events. Figure 2 shows the event model for a periodic event arrival with a period of 250 time units and a jitter of 250 time units. The jitter results in the fact that two events can arrive simultaneously ($\delta^-(2) = 0$) but also up to two periods apart ($\delta^+(2) = 500$).

To obtain the worst-case timing of a system, CPA first performs so-called local "busy window" analyses on each resource to compute the worst-case timing and output event model of each task. For this, a critical instant scenario is constructed, assuming the worst-case arrival of all interfering tasks to maximally delay the processing of the task under consideration. After all local analyses, the output event models of each task is forwarded as input event models of the dependent tasks, which are then analyzed again using the updated event models. This procedure is iterated until a fixed

point (stable event models) is reached or a timing constraint (e.g. maximum path latency) is violated [15]. Initial event models for all tasks are derived from the external input event model of each task chain. In addition to the validation of the schedulability (all deadlines met) the analysis also yields upper bounds on the worst-case response time (WCRT) of tasks and other timing properties such as the worst-case end-to-end path latency of a chain of tasks.

III. Analysis

In this section, we present the analysis of strict-priority Ethernet and Ethernet AVB using the CPA approach. Most of the discussion will focus on latter Ethernet standard, as from the timing perspective, strict-priority Ethernet is just a special case of Ethernet AVB with disabled traffic shaping. Also, for simpler presentation, we assume that every traffic class can be shaped by a traffic shaper. For classes not having a traffic shaper, the $idleSlope$ can be set to the port transmission rate which makes the shaper ineffective.

To apply CPA to Ethernet networks, it is required to map the Ethernet components to the modeling artifacts of CPA as shown in [18], [7]. Since we want to analyze the arbitration at the output ports of Ethernet switches, it is natural to map them to processing resources in CPA. For every traffic stream (i.e. talker-listener pair with unique traffic characteristics) we add a dedicated task to each output port it passes through. The core execution time of the tasks matches the transfer time of the Ethernet frames of that stream. The arrival of a frame then becomes the activation event of a task. A traffic stream for a source to a destination over multiple hops then becomes a chain of tasks mapped to a series of resources modeling the switch outputs. For such a chain, we can compute the worst-case path latency which is equivalent to the transfer latency of the Ethernet stream. For the remainder of the paper, we will mostly use the terms from the CPA domain.

Now being able to map the analysis of Ethernet AVB timing to the performance analysis according to CPA, we need to derive a local analysis, which computes the worst-case response-time of a task and the corresponding output event model. In CPA, the local analysis requires the formulation of a level-i busy-time $B_i^+(q)$ (cf. busy-period in [22]) which describes how long a resource is busy processing q jobs of task τ_i. For AVB, we need to extend the definition of the busy-period to account for the traffic shaper:

Definition 1. *The maximum (minimum) q-event busy-time $B_i^+(q)$ ($B_i^-(q)$) of a task τ_i is given by the maximum (minimum) time the resource is busy processing q events, if all but the first of the q events arrive within the busy-time of their respective predecessor. The resource is considered busy if it processes a task or if the traffic shaper corresponding to task τ_i still has negative credit.*

In order to conservatively capture all possible effects, we break down the busy-time into a sum of different terms, each addressing an individual effect. Under AVB scheduling, the busy-time of a task τ_i is impacted by:

- Transfer time $t_{transfer}$: The time to execute the task τ_i (transfer a frame) is determined by the maximum core execution time not including any blocking (no-load transfer time). The maximum core execution time results from the network speed and frame size, and is the maximum time required to transfer a frame through the corresponding port.
- Blocking time by lower-priority tasks I_{LPB}: Task τ_i can be blocked by one lower-priority task activation that commenced transfer just before the activation of the task.
- Blocking time by same-priority frames I_{SPB}: Task τ_i can be blocked by other tasks of the same priority which were activated before itself.
- Blocking time by higher-priority frames I_{HPB}: All higher-priority tasks may block task τ_i, limited by the traffic shaping applied to the high-priority classes.
- Blocking time by traffic shaping I_{TSB}: Task τ_i may have to wait for shaper credits to be replenished before it may execute. This is omitted for strict-priority Ethernet.

In order to obtain the maximum busy-time $B_i^+(q)$, i.e. the longest time required to execute q activations of a task τ_i (which equals the transfer of q frames of the corresponding stream), we maximize all of the above delays and add them up:

$$\begin{aligned} B_i^+(q, a_i^q) \leq & t_{transfer}(q) + I_{LPB} + I_{SPB}(a_i^q) + \\ & I_{TSB}(a_i^q) + I_{HPB}(B_i^+(q, a_i^q)) \end{aligned} \tag{1}$$

where a_i^q is the arrival time of the q-th activation of task τ_i relative to the beginning of the busy time.

Note that the same-priority blocking and the traffic-shaper blocking depend on the arrival-time a_i^q of the q-th activation, whereas the higher-priority blocking depends on the busy-time $B_i^+(q, a_i^q)$ itself. Because B_i^+ appears on both sides of the equation, it forms an integer fixed point problem, which is typical for busy-time based scheduling analysis. It can be resolved iteratively by starting with $B_i^+(q, a_i^q) = t_{transfer}(q) + I_{LPB}$.

A. Individual Sources of Blocking

We will now discuss the upper bounds of each component of $B_i^+(q, a_i^q)$. The **maximum transfer time** for q activations of a task τ_i is given by

$$t_{transfer}(q) = q \cdot C_i^+ \tag{2}$$

The worst-case execution time C_i^+ of the task τ_i is equal to the maximum time it takes for a frame of the corresponding stream to pass through the arbitration point without contention. Respecting the minimum Ethernet frame size, the frame overhead and the inter-frame gap, C_i^+ can be obtained as

$$C_i^+ = \frac{(48Byte + max(36Byte, dataLength))}{portTransmissionRate} \tag{3}$$

For fixed frame sizes, the minimum execution time C_i^- is equal to the maximum execution time C_i^+.

A task τ_i can suffer **lower-priority blocking** only once by a non-preemptive lower-priority task that started executing just before τ_i was ready. In the worst case, the longest executing lower-priority task must be assumed to be the blocker:

$$I_{LPB} = \max_{j \in lp(i)} \left\{ C_j^+ \right\} \tag{4}$$

where $lp(i)$ is the set of lower priority tasks mapped to the same resource as task τ_i.

The **same-priority blocking** depends on the arrival time a_i^q of the q-th event of task τ_i due to the FIFO scheduling within the same priority. Hence, an upper bound for the blocking inferred by the same-priority tasks is

$$I_{SPB}(a_i^q) = \sum_{j \in sp(i)} \left(\eta_j^+(a_i^q) \cdot C_j^+ \right) \tag{5}$$

with $sp(i)$ being the set of same-priority tasks mapped to the same resource as task τ_i and η_j^+ being the worst-case arrival function of task τ_j. Thus $\eta_j^+(a_i^q)$ is the maximum number of activations of task τ_j before the arrival of the q-th activation of task τ_i. As an upper bound for the worst-case, we assume that all of these activations block task τ_i by the worst-case execution time C_j^+ of the corresponding task τ_j.

For the **higher-priority blocking**, we have to assume in the worst-case that all higher priority activations arriving just before the transmission of task τ_i interfere with it. Hence, we obtain

$$I_{HPB}(B_i^+(q, a_i^q)) = \sum_{j \in hp(i)} \left(\eta_j^+(B_i^+(q, a_i^q) - C_i^+) \cdot C_j^+ \right) \tag{6}$$

with $hp(i)$ being the set of higher-priority tasks mapped to the same resource as task τ_i and η_j^+ being the worst-case arrival function of task τ_j. Thus $\eta_j^+(B_i^+(q, a_i^q) - C_i^+)$ is the maximum number of activations of task τ_j before the execution of the q-th activation of task τ_i. Note that we subtract C_i^+ from $B_i^+(q, a_i^q)$ because arrivals of higher-priority tasks can not interfere once the q-th activation of task τ_i has started executing due to the non-preemptiveness of Ethernet transfers.

The bounds of the same- and higher-priority interference can be reduced by exploiting the traffic shaping on preceding resources, which will be described in Section III-C.

The **traffic shaper blocking** is interdependent with other interference because positive credit accumulates when a task is blocked which reduces traffic shaper blocking later. Hence, in order to maximize the overall interference, i.e. the sum of the terms in Equation 1, we assume that the traffic shaper interference occurs as early as possible (by assuming no positive credit), according to the following lemma.

Lemma 1. *While a task τ_i is blocked by a task τ_j of a different priority, any blocking by a traffic shaper afterwards is delayed.*

Proof: Assume task τ_j blocks tasks τ_i by exactly t_{block} time. During this time, the credit of task τ_i's shaper increases at the $idleSlope$ by $c_{block} = t_{block} \cdot idleSlope$. This credit allows task τ_i to execute for an extra time of $c_{block} \cdot (-sendSlope)$ before the traffic shaper depletes. From this, the lemma follows. ∎

The **traffic shaper blocking** depends on the allowed rate ($idleSlope_{c(i)}$) of the class $c(i)$ of task τ_i. Every task executed on the class of task τ_i (i.e. tasks in $sp(i) \cup \{\tau_i\}$) consumes credits, which may lead to a blocking by the shaper. The amount of credits consumed by executing C_{trans} time units is $K_{consumed} = -sendSlope_{c(i)} \cdot C_{trans}$. Considering traffic shaper blocking observed by task τ_i, C_{trans} can be bounded by the time required to transmit own and same-priority frames, i.e. $C_{trans} \leq t_{transfer}(q) + I_{SPB}(a_i^q)$. Hence, to replenish the credits consumed by previous frame transfers, the following time is required:

$$I_{TSB}(a_i^q) = \frac{K_{consumed}}{idleSlope_{c(i)}} = \frac{-sendSlope_{c(i)}}{idleSlope_{c(i)}} \cdot C_{trans} \tag{7}$$

$$\leq \left[(q-1) \cdot C_i^+ + I_{SPB}(a_i^q) \right] \cdot \frac{-sendSlope_{c(i)}}{idleSlope_{c(i)}} \tag{8}$$

This formula assumes that the first frame does not observe any traffic shaper blocking according to the definition of the busy-time (Definition 1), hence the $q-1$. Note again that for strict-priority Ethernet, $I_{TSB} = 0$.

B. Worst-Case Response Time and Output Event Model

Now that we have derived the q-event busy-time for Ethernet AVB scheduling, we can bound the worst-case response time R_i^+ of task τ_i, which is equivalent to the maximum per-hop frame latency. For this, we must find the maximum distance between the completion of q activations ($B_i^+(q)$) and the arrival of the q-th activation a_i^q:

$$R_i^+ = \max_{q \in Q_i} \left\{ \max_{a_i^q \in A_i} \left\{ B_i^+(q, a_i^q) - a_i^q \right\} \right\} \tag{9}$$

This equation forms a nested optimization problem: for every number of activations q within a busy-time, all possible arrival times a_i^q of the q-th event have to be considered to find the maximum response time. Considering different arrival times is necessary because a later arrival increases the interference due to FIFO scheduling but at the same time reduces the response time. Fortunately, the number of possible candidates for a_i^q is finite. One has to only consider activations of task τ_i that happen just after the activation of other same-priority tasks, which are defined by their respective δ^--functions. An arrival later than just after a same-priority activation but before the arrival of another same-priority activation would not increase the load (as no additional events arrive) but would decrease the response time, as the arrival time is subtracted from the busy time (see Equation 9). Hence, the set A_i^q) of candidates for a_i^q within the busy-time $B_i^+(q, a_i^q)$ can be written as:

$$A_i^q = \bigcup_{j \in sp(i)} \left\{ \delta_j^-(n) | n > 0 \wedge \delta_j^-(n) < B_i^+(q, a_i^q) \right\} \quad (10)$$

Note that this set again includes a fixed-point iteration as $B_i^+(q, a_i^q)$ is required to compute the set. For the number of activations q within a busy-time, we need to consider all scenarios where an activation arrives within the busy-time of the previous activation, i.e. all $q \geq 1$ which are smaller than the maximum number of events q_i^+ which fall into one busy-time (see [23]):

$$Q_i = \left\{1, 2, \ldots, q_i^+\right\} \quad (11)$$

$$q_i^+ = \min\{q \in \mathbb{N}^+ | \delta_i^-(q+1) \geq B_i^+(q, a_i^q = 0)\} \quad (12)$$

We can derive the output event model for each task which becomes the input event models of dependent tasks as proposed in [24] (refer to the paper for further details on the equations):

$$\delta_{i,out}^-(n) = \max\Big\{ B_i^-(n-1), \min_{j \in Q_i} \{\delta_{i,in}^-(n+j-1) - B_i^+(j)\} + B_i^-(1) \Big\} \quad (13)$$

$$\delta_{o,out}^+(n) = \max_{j \in Q_i} \Big\{ \delta_{i,in}^+(n-j+1) + B_i^+(j) \Big\} - B_i^-(1) \quad (14)$$

Here, $B_i^-(q)$ is the minimum busy-time which describes the smallest time window a resource is busy processing q events. Trivially, this is q times the best-case execution time C_i^-:

$$B_i^-(q) = q \cdot C_i^- \quad (15)$$

This can be improved using knowledge about the traffic shaper as we will discuss in the next section. The equations for the output event model are used during the event model propagation step after the local analyses.

The worst-case response time R_i^+ of task τ_i is equivalent to the maximum delay any frame of the corresponding stream will observe on the corresponding hop. The end-to-end latency $l_p^+(s)$ of the transferal of s frames on stream p can be derived from the sum of the per-hop delays and the additional wire and (de-)packetization delays:

$$l_p^+(s) \leq \delta_{First(p)}^-(s) + \sum_{j \in Tasks(p)} R_j^+ + O_{routing}(p) \quad (16)$$

where $First(p)$ denotes the first task (source) stream p, $Tasks(p)$ denotes the set of all tasks (one per hop) of stream p, and $O_{routing}(p)$ denotes the constant total routing/wire overhead of stream p (which can be a function of the number of hops). This equation basically computes how long it takes to inject s frames at the source node ($\delta_{First(p)}^-(s)$) and then assumes that the last one of these will observe the worst-case response time on all hops. Due to in-order delivery, all previous frames will also have reached the destination before the last one. The delay that these previous frames observe is included as interference in the worst-case response time of the last frame.

CPA can also be used to derive the maximum required buffer size to avoid frame drops due to buffer overflows. For this, we simply derive the maximum activation backlog b_i^+, i.e. the maximum number of activations of task τ_i which have arrived but not yet processed. It can be computed as

$$b_i^+ = \max_{q \in Q_i} \{\eta_i^+(B_i^+(q)) - (q-1)\} \quad (17)$$

This equation examines all numbers of activations q which fall into the same busy time. For each, $B_i^+(q)$ yields the completion time of the q-th activation. At this instance of time, at most $\eta_i^+(B_i^+(q))$ activations have arrived since the beginning of the busy time, of which $q-1$ have already been processed. Hence, the difference yields the amount of outstanding activations, of which we have to determine the maximum for all $q \in Q_i$.

C. Improving the Interference Bounds by Exploiting the Traffic Shaping

So far, we have used the traffic shaping only in I_{TSB} introducing additional delay to frames. However, the traffic shaping also improves the timing for dependent tasks as it forces delays between events. This reduction in burstiness reduces the interference on subsequent resources.

Intuitively, this should be captured during the output event model propagation where the event model at the output of each router (i.e. after traffic shaping) is computed (see Equations 13 and 14). However, we can only partly exploit the traffic shaper here as the shaper works per-class and not per-stream. Still, we can improve the output event model slightly under the conservative assumption that all credit available in the shaper is used by just one stream. We express the delay imposed by the traffic shaping in the output event model propagation (Equations 13 and 14) by an improved minimum busy-time $B_i^-(q, a_i^q)$ describing the smallest time window a resource is busy processing n events. The improved minimum busy-time is

$$B_{i,shaping}^-(q) = q \cdot C_i^- + I_{i,TSB}^-(q) \quad (18)$$

where $I_{i,TSB}^-(q)$ is the minimum guaranteed blocking inferred by the shaper to task τ_i.

Figure 3 illustrates the minimum time $I_{i,TSB}^-(q)$ that q activations of task τ_i are delayed by the traffic shaper. The dashed line shows the credit of the traffic shaper assuming the credit is initially zero. As the $q-1$ events prior to the q-th activation are executed, the credit level continually reduces and needs to recover before the q-th activation may execute. Note that the figure shows that all $q-1$ activations are executed at once, but in reality, the traffic shaper blocks task τ_i after every execution. For the total blocking time, however, this does not make any difference. There are no

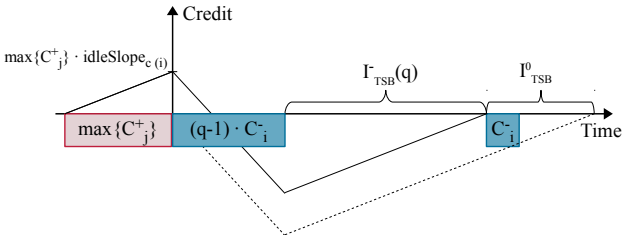

Fig. 3. Example for the computation of the minimum blocking of the traffic shaper $I^-_{i,TSB}(q)$.

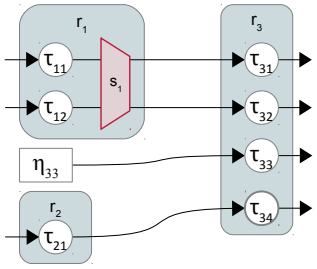

Fig. 4. Illustration of the exploitation of the traffic shaper.

other same-priority interferers of τ_i executing which means that all credit is available to task τ_i, minimizing the blocking. This results in a conservative lower bound on the minimum busy-time. Note that the presence of other streams could be exploited to reduce the credits available to each stream and hence increase the minimum delays imposed by the shaper. Furthermore, we need to conservatively assume that the traffic shaper credit level is not initially zero due to a prior interfering frame of tasks of other priority as illustrated in Figure 3. The solid line shows the credit level under the presence of such an interferer, and it can be seen that it reaches zero I^0_{TSB} time units before the dashed line. I^0_{TSB} is the duration of an initial burst allowed by the shaper due to positive credit. Hence, the actual minimum distance time needs to be reduced by this amount. With these considerations, we can express the minimum time that q activations are delayed by the traffic shaper as

$$I^-_{i,TSB}(q) = \max\Big\{0,(q-1)\cdot C^-_i \cdot \frac{-sendSlope_{c(i)}}{idleSlope_{c(i)}} - I^0_{TSB}\Big\} \tag{19}$$

The initial burst I^0_{TSB} can be computed from the maximum execution time of any interfering task:

$$I^0_{TSB} = \max_{j\in lp(i)\cup hp(i)} \{C^+_j\} \cdot \frac{idleSlope_{c(i)}}{-sendSlope_{c(i)}} \tag{20}$$

As discussed above, the traffic shaper blocking can only partly be exploited in the event model propagation because the shaper works per class and not per task. To consider the per-class shaping, we can extend the computation of the same-priority blocking.

For illustration, take a look at Figure 4 which shows three resources r_1, r_2, and r_3 with multiple tasks of which we assume all have the same priority. Let us consider the same-priority interference of task τ_{34} on resource r_3. The interfering tasks τ_{31} and τ_{32} both are activated from tasks τ_{11} and τ_{12} running on the same resource r_1 and being shaped by the same traffic shaper s_1. So far, we have assumed that both tasks τ_{31} and τ_{32} may be activated simultaneously and both interfere with task τ_{34}. However, this can not be the case as the previous resource can only process and hence finish one task at a time (due to non-preemptiveness). Hence, we can group same-priority interferers by the resource from which they are activated and then provide an upper bound for the combined load from each group of tasks. This works even without traffic shaping, as the processing speed of the resource is finite, but with exploiting traffic shaping we can give a tighter bound on the load. Of course, this does not work for tasks which are not activated from a previous resource, such as task τ_{33} in the example which is activated from an external event model η_{33}. For such tasks, we calculate the interference as shown in Equation 5.

In [25], a similar approach has been presented for resources with preemptive scheduling which considered minimum distances between events of different tasks activated from the same resource. This allows the analysis of task chains with different execution times on each resource, which is not needed for Ethernet so we can use a simpler load limit in this paper.

With the above reasoning, we first compute the maximum load $L_s(\Delta t)$ that is allowed by a shaper s within any time window Δt

$$L_s(\Delta t) = \min\Big\{\Delta t, I^0_{TSB} + \Delta t \cdot \frac{idleSlope_s}{-sendSlope_s}\Big\} + \max_{k\in tasks(s)}\{C^+_k\} \tag{21}$$

where $tasks(s)$ is the set of tasks shaped by the shaper s.

In this equation, the first term in the min-function describes an upper bound to the load that the port allows without any shaping within Δt time. The second term in the min-function gives an upper bound on the shaped load, again including the maximum initial workload (see Equation 20) similarly to Equation 19, The last term in Equation 21 accounts for the fact that in the worst case, another maximum-size same-priority interferer may be allowed by the shaper if the credit is just above zero.

Now, we need to group tasks by their preceding resource. For this, we define R_i to be the set of resources from which same-priority interferers of task τ_i are activated. In the example $R_{34} = \{r_1\}$. We define $sp_r(i)$ to be the set of same-priority interferers of task τ_i which are activated from resource r and s^r_i to be the shaper on resource r for the same class of task τ_i. So in the example, $sp_{r_1}(34) = \{\tau_{31}, \tau_{32}\}$ and

$s_{34}^{r_1} = s_1$. Furthermore, we define $np(i)$ as the set of same-priority tasks of task τ_i which are not activated from a task on a different resource (in the example $np(34) = \{\tau_{33}\}$). From this, we can now improve the same-priority blocking:

$$I_{SPB}^{shaped}(a_i^q) = \sum_{r \in R_i} \min\Big\{ L_{s_i^r}(a_i^q), \sum_{j \in sp_r(i)} \eta_j^+(a_i^q) \cdot C_j \Big\} + \sum_{j \in np(i)} \eta_j^+(a_i^q) \cdot C_j \quad (22)$$

For all tasks which are activated from traffic shaped resources, this equation limits their accumulated load to the maximum load allowed by the traffic shaper (first sum). For all tasks that are not activated from a shaped task, the equation simply computes the maximum interference as before in Equation 5. Note that with this extension, the candidates for the arrivals a_i^q need to be extended to cover all arrivals during which the load of the same-priority interferers is still being shaped. For this, we iteratively add the times at which the current load is no longer shaped to the arrival candidates.

Likewise, we can extend the higher-priority blocking to account for shaping on previous resources to reduce the interference of shaped traffic on lower-priority traffic.

$$I_{HPB}^{shaped}(B_i^+(q, a_i^q)) = \sum_{p \in prio(hp(i))} \min\Big\{ L_{s_p}(B_i^+(q, a_i^q)), \sum_{j \in tasks(p)} \eta_j^+(B_i^+(q, a_i^q)) \cdot C_j \Big\} \quad (23)$$

where $prio(hp(i))$ is the set of higher priorities of task τ_i, s_p is the shaper of priority p on the resource of task τ_i, and $tasks(p)$ are the tasks in priority p on the resource of task τ_i. Similarly to the improved same-priority blocking in Equation 22, we group tasks to consider the limit imposed by the shaper on the task group. For the higher-priority blocking, we group by priority (instead of by preceding resource) and apply the maximum load L_{s_p} (Equation 21) allowed by the shaper of this priority as an upper bound to the interference.

IV. Evaluation of Different Topologies

Now that we have presented the analysis approach, we will apply this to evaluate different topologies for a typical use case from industrial automation. For the use-case, we assume a network with a single controller which periodically sends and receives control data to multiple I/O nodes. Specifically, we will evaluate a star and a line topology, as shown in Figure 5. For every topology, we will compare the worst-case timing of Ethernet AVB to strict-priority (SP). Both controller and nodes are assumed to send with the same period of 250 μs with a jitter of 250 μs, which corresponds to the event model shown in Figure 2. This jitter means that there can be a burst of two activations. The message size is assumed to be 10 Bytes, which fits into a minimum-size Ethernet frame. If not stated otherwise, we assume that all control traffic (in both directions) is sent on Class A for AVB or priority 3 for strict-priority Ethernet. For all topologies, we assume a wire delay of 330 ns. For AVB, the $idleSlope$ is usually set up to allow just enough credit required to sustain the accumulated rate of all traffic streams. However, a transient overload (introduced by jitter) results in additional frames which may keep the buffers occupied forever if the traffic shapers allow no extra load. This "permanent delay" is discussed in the IEEE 802.1Qav standard [5] and the standard suggests a "little" over-reservation to cope with this. Hence, we multiply all $idleSlopes$ by a over-reservation factor of 2 if not stated otherwise. This means that technically, we reserve twice as much bandwidth for class A than is injected, to accommodate the bursts of two activations of each stream.

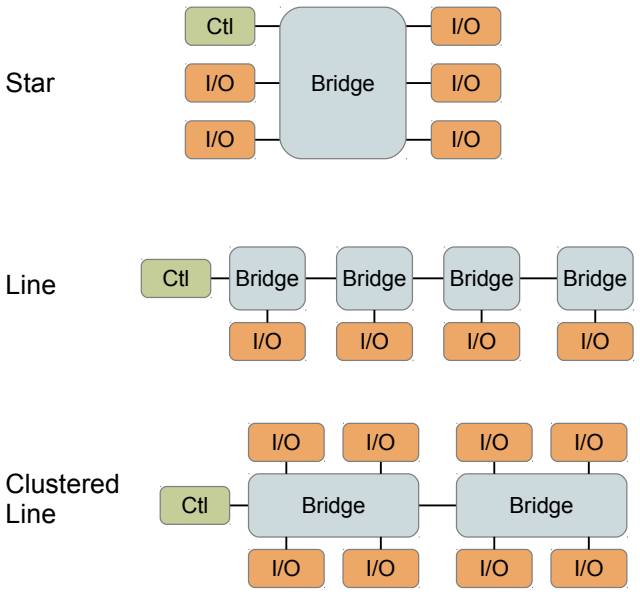

Fig. 5. Topologies of the evaluation.

In all experiments, we assume that there is unknown non-real-time traffic (NRT). This means that independent lower-priority blockers of the size of a maximum Ethernet frame must be considered at each hop. Note that in the system graphs, we do not show the corresponding NRT tasks.

A. Star Topology

We will first evaluate a star topology. Here, the controller and all devices are connected to a single central Ethernet bridge. We will vary the number of I/O nodes and hence ports of the bridge. From the analysis point-of-view, this is the most straight-forward topology, as all interference happens only at one place. Specifically, the output port connecting the switch to the controller will see interference from all nodes sending control data. The other direction, from the controller to the nodes, is contention-free, as each node is connected directly via a dedicated port.

Figure 6 shows the system model of a star-topology for three I/O nodes. Hence, the central switch ("S0") has four

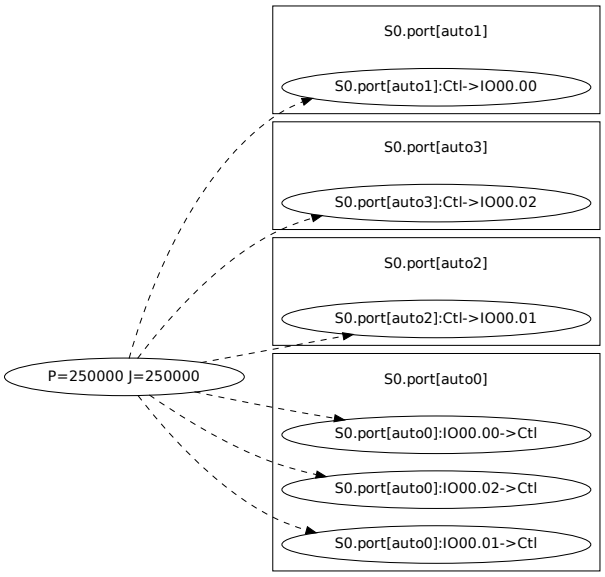

Fig. 6. System model of a star topology with 1 controller and 3 I/O nodes.

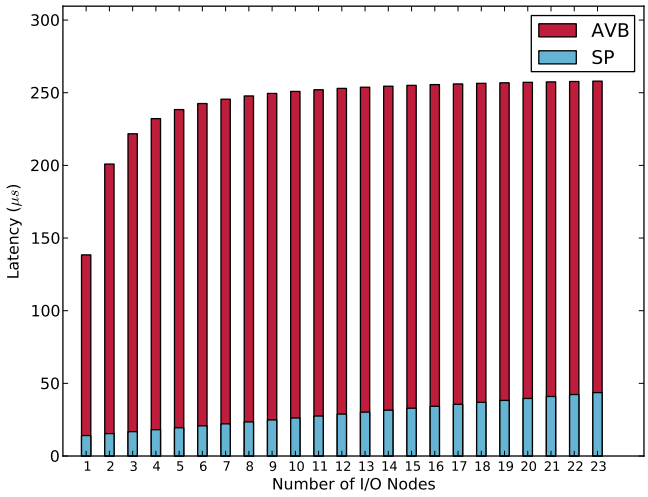

Fig. 7. Worst-case communication latencies to the controller for star topologies with different numbers of nodes.

ports (three for I/O, one for the controller). The latencies for communication between the nodes and the controller do not depend on the switching protocol (AVB vs strict-priority) nor on the number of connected nodes. It is $1002\ \mu s$ including $330\ ns$ wire delay, which is the best-case transfer time.

For the other direction, Figure 7 shows the worst-case latencies for Ethernet AVB and strict-priority for up to 23 I/O nodes. The largest configuration can be implemented with a standard-sized 24-port bridge. As expected, the latency for strict-priority Ethernet linearly increases with the number of nodes due to the increasing same-priority interference. For Ethernet AVB, the plot shows much higher latencies than for strict-priority Ethernet. This is due to the traffic

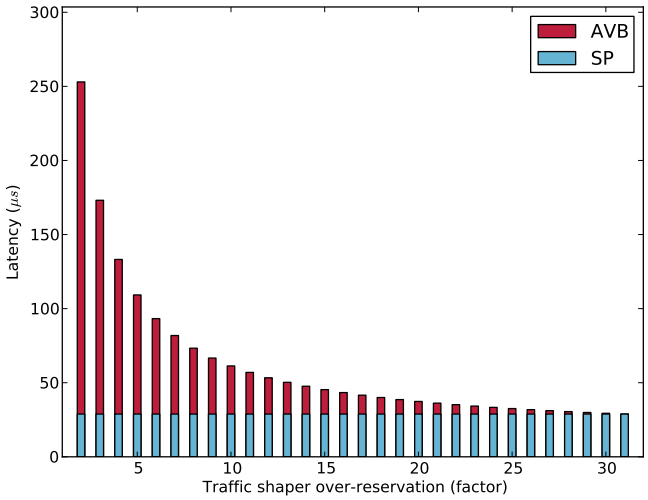

Fig. 8. Worst-case communication latencies to the controller for the star topology with 12 nodes and different factors for over-reservation of the traffic shapers.

shaping applied to class A, which delays frames which arrive bursty (which is assumed for the worst-case). Furthermore, the latencies of AVB do not rise linearly, but following a function of the shape of $(n-1)/n$ where n is the number of nodes. This results from the fact that although the same-priority and traffic-shaper interference increase with $n-1$, the $idleSlope$ increases with n. Hence, the traffic-shaper blocking grows super-linearly for small numbers of nodes and linearly for large numbers of nodes.

As discussed above, the $idleSlope$ for class A traffic has been configured to allow twice as much as the required bandwidth to process transient overloads quickly. To see the effect of this over-reservation, we keep the number of nodes constant at 12 but vary the factor of the over-reservation from 2 to 31. Figure 8 shows the resulting latencies for the traffic to the controller. It can be seen that the latency for AVB drops with increasing over-reservation. As expected, it approaches the latency of strict-priority Ethernet as the impact of the traffic shaper is reduced. In our example, both switching schemes are equivalent with the $idleSlope$ set to allow 31x the required bandwidth.

We have seen that AVB has a serious impact on the worst-case latency due to the blocking by the traffic shaper. However, we would expect that the blocking of class A improves the latency of class B. To evaluate this hypothesis, we add a class B stream to each of the I/O nodes which behaves exactly as the class A stream except for the period which is raised to $2.5\ ms$ (10x more). Here, we examine only the case with 23 I/O nodes. With strict-priority scheduling, the latency of the class B traffic to the controller is $61\ \mu s$, which is about 40% more than the latency of the class A frames. We set up the $idleSlope$ for class B the same way as for class A, i.e. to two-times the requested bandwidth. With AVB, however, the class B latency amounts to $1.3\ ms$, which is more than 5x the latency of class

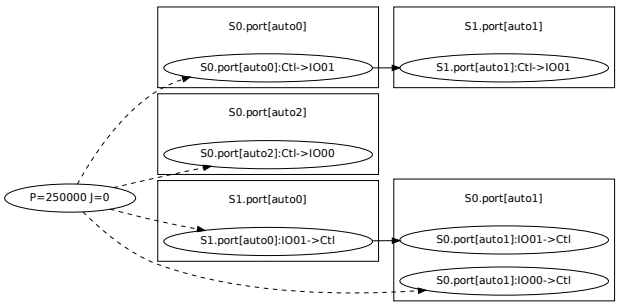

Fig. 9. System model of a line topology.

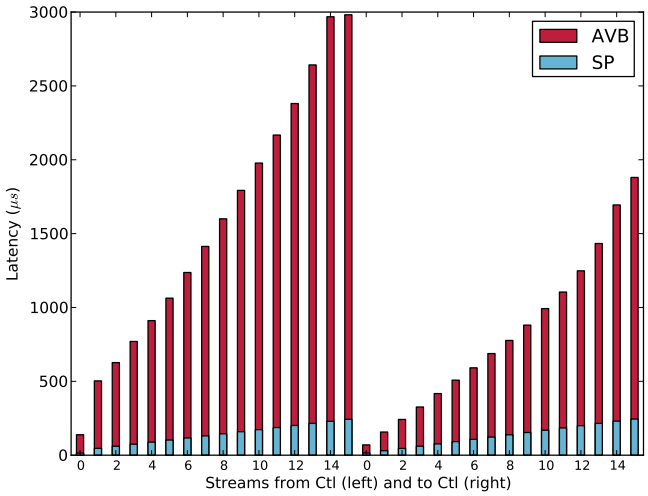

Fig. 10. Worst-case communication latencies for the streams in the linear topology.

A traffic. This is due to the fact that class B itself is traffic shaped, which means that in the worst-case (in this example) it can not use the slots during which class A is shaped because it is blocked by its own shaper. If we disable the traffic shaping of class B, its latency drops to 35 μs, which is lower than the latency in the strict-priority case. This shows that the shaping of AVB on class A does not improve the worst-case latency of class B, but only that of lower-priority unshaped traffic. With other traffic constellations a positive effect of the shaping of class A on class B can be observed: if we switch the periods of classes A and B, the worst-case latency of class B increases from 262 μs to 274 μs once we disable the shaping on class A. This positive effect is not enough to counteract the delay of the shaping on class B itself compared to the strict-priority case. However, do note that the shaping of class A (and B) allows a traffic-independent guarantee of a minimum bandwidth to the traffic on class B (and lower priorities). These considerations remain valid for the other topologies, but we will focus only on class A traffic there.

B. Linear Topology

We will now evaluate a linear topology, where each node is connected via a dedicated 3-port bridge to two neighbors. As opposed to the previous star topology, a linear topology results in much more interference, as traffic needs to pass the same output ports along the line of bridges. Hence we expect much more interference.

Figure 9 shows the system model for a line topology with two switches S0 and S1. To each switch, an I/O node (IO00 and IO01) is connected, and the controller "Ctl" is connected to switch S0. The figure also shows the task chains modeling the communications between the controller and the I/O nodes.

For this topology, we keep the number of switches and nodes constant to 16 but evaluate the latencies for the communication to/from each node. Figure 10 shows the latencies for the worst-case for both AVB and strict-priority Ethernet for the different streams. The left half of the plot shows the traffic streams from the controller to the I/O nodes, while the right half shows the other direction. Similarly to the star topology, it can be seen that the worst-case latencies for strict-priority Ethernet increase linearly with the distance between the controller and the node. The same is true for the latency of AVB traffic, which is in contrast to the star topology. Also, AVB latencies grow at a much higher slope. The reason for this is that in a linear topology, many streams run in parallel for multiple hops. The traffic shapers at each hop induce jitters, which leads to increased bursts at subsequent resources, so the impact of the traffic shaper accumulates. Note that the jitter grows so high that much longer lines are not reasonable with this traffic setup.

C. Clustered Linear Topology

Finally, we evaluate a clustered linear topology, in which multiple star topologies are connected in a line. Figure 11 shows the latencies for a clustered line with 4 bridges with 6 I/O nodes each, which can be constructed from 8-port bridges. The controller is connected to the first bridge. The behavior is as expected a mix of the star and the line topology. One can observe 4 groups of 6 streams with the same latencies, with the latencies of the individual groups increasing about linearly. Again, AVB yields much higher latencies than strict-priority Ethernet. As expected, the latencies in the clustered topology are between those of the star topology and the line topology.

V. Conclusion

In this paper, we have presented a formal worst-case analysis for Ethernet networks using strict-priority arbitration and the credit-based-shaping-algorithm from Ethernet AVB. These allow the derivation of upper bounds on the timing of frame transfers, which are required to employ Ethernet networks in real-time and safety-critical embedded systems. Using this approach, we evaluated several topologies for a typical use-case in industrial automation. The evaluation has generally shown that Ethernet AVB first of all substantially increases the latencies for the highest-priority Class-A traffic compared to the static-priority arbitration due to the additional traffic shaping delay. We have also seen in the evaluation that class B traffic can not benefit from the shaping of class A

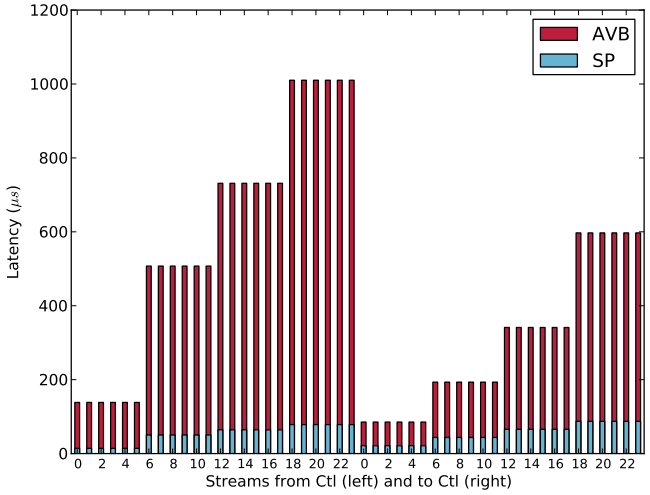

Fig. 11. Latencies for the streams in the clustered topology.

as it is shaped itself and hence in the worst-case can not utilize the time slots during which class A traffic is shaped. However, the worst-case latency of unshaped lower priorities does improve due to the shaping of classes A and B. Also, the shaping allows to give minimum bandwidth guarantees to lower priority traffic.

REFERENCES

[1] R. Cummings, K. Richter, R. Ernst, J. Diemer, and A. Ghosal, "Exploring use of ethernet for in-vehicle control applications: Afdx, ttethernet, ethercat, and avb," in *SAE 2012 World Congress & Exhibition*, 4 2012, SAE Technical Paper 2012-01-0196. [Online]. Available: http://papers.sae.org/2012-01-0196

[2] "Audio/Video Bridging Task Group of IEEE 802.1." [Online]. Available: http://www.ieee802.org/1/pages/avbridges.html

[3] "IEEE Standard 802.1AS-2011 – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks," 2011. [Online]. Available: http://standards.ieee.org/findstds/standard/802.1AS-2011.html

[4] "IEEE Standard 802.1Qat-2010 – Stream Reservation Protocol (SRP)," 2010. [Online]. Available: http://standards.ieee.org/findstds/standard/802.1Qat-2010.html

[5] "IEEE Standard 802.1Qav-2009 – Forwarding and Queuing Enhancements for Time-Sensitive Streams," 2009. [Online]. Available: http://standards.ieee.org/findstds/standard/802.1Qav-2009.html

[6] "IEEE Standard 802.1Q-2011 – IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks," 2011. [Online]. Available: http://standards.ieee.org/findstds/standard/802.1Q-2011.html

[7] J. Diemer, J. Rox, and R. Ernst, "Modeling of Ethernet AVB Networks for Worst-Case Timing Analysis," in *MATHMOD*, Austria, 2012.

[8] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised," *Real-Time Systems*, vol. 35, no. 3, 2007.

[9] J. Palencia and M. Harbour, "Offset-based response time analysis of distributed systems scheduled under EDF," in *Real-Time Systems, 2003. Proceedings. 15th Euromicro Conference on*, 2003.

[10] K. C. Lee, S. Lee, and M. H. Lee, "Worst case communication delay of real-time industrial switched ethernet with multiple levels," *Industrial Electronics, IEEE Transactions on*, vol. 53, no. 5, pp. 1669 –1676, oct. 2006.

[11] K. Revsbech, H. Schi, T. Madsen, and J. Nielsen, "Worst-case traversal time modelling of ethernet based in-car networks using real time calculus," in *Lecture Notes in Computer Science*. Springer, 2011.

[12] J.-P. Georges, T. Divoux, and E. Rondeau, "Strict priority versus weighted fair queueing in switched ethernet networks for time critical applications," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, april 2005, p. 141.

[13] J. Imtiaz, J. Jasperneite, and L. Han, "A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, 2009.

[14] K. Richter, M. Jersak, and R. Ernst, "A Formal Approach to MpSoC Performance Verification," *IEEE Computer*, vol. 36, no. 4, apr 2003.

[15] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis–the SymTA/S Approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, 2005.

[16] S. Schliecker, J. Rox, M. Negrean, K. Richter, M. Jersak, and R. Ernst, "System Level Performance Analysis for Real-Time Automotive Multi-Core and Network Architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2009.

[17] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *ISCAS*, vol. 4, 2000.

[18] J. Rox and R. Ernst, "Formal Timing Analysis of Full Duplex Switched Based Ethernet Network Architectures," in *SAE World Congress*, vol. System Level Architecture Design Tools and Methods (AE318). SAE International, Apr 2010.

[19] Z. Shi and A. Burns, "Real-time communication analysis for on-chip networks with wormhole switching," in *NOCS*. IEEE Computer Society, 2008.

[20] J. Diemer, J. Rox, M. Negrean, S. Stein, and R. Ernst, "Real-Time Communication Analysis for Networks with Two-Stage Arbitration," in *EMSOFT'11*, October 2011.

[21] Jonas Diemer and Philip Axer, "pyCPA: Compositional Performance Analysis in Python," http://code.google.com/p/pycpa/, 2012.

[22] K. Tindell, A. Burns, and A. Wellings, "An extendible approach for analyzing fixed priority hard real-time tasks," *Real-Time Systems*, vol. 6, no. 2, 1994.

[23] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *Design, Automation and Test in Europe*, 2012.

[24] S. Schliecker, J. Rox, M. Ivers, and R. Ernst, "Providing Accurate Event Models for the Analysis of Heterogeneous Multiprocessor Systems," in *CODES-ISSS*, oct 2008.

[25] J. Rox and R. Ernst, "Exploiting inter-event stream correlations between output event streams of non-preemptively scheduled tasks," in *Proc. Design, Automation and Test in Europe (DATE 2010)*, March 2010.