Timing Analysis of Rate Constrained Traffic for the TTEthernet Communication Protocol

Domiţian Tămaş–Selicean and Paul Pop
Technical University of Denmark
DTU Compute
Kongens Lyngby, 2800, Denmark
{dota,paupo}@dtu.dk

Wilfried Steiner
TTTech Computertechnik AG
Vienna, Austria
wilfried.steiner@tttech.com

***Abstract*—Ethernet is a low-cost communication solution offering high transmission speeds. Although its applications extend beyond computer networking, Ethernet is not suitable for real-time and safety-critical systems. To alleviate this, several real-time Ethernet-based communication protocols have been proposed, such as TTEthernet, which is the focus of this paper. TTEthernet is suitable for mixed-criticality systems both in the safety and temporal domain. TTEthernet offers three traffic classes: static time-triggered (TT) traffic, dynamic traffic with bounded transmission rate (called "Rate Constrained", RC), and unbounded dynamic traffic (BE). In this paper we propose a novel worst-case end-to-end delay analysis of the RC traffic for TTEthernet systems. The proposed technique considerably reduces the pessimism of the analysis, compared to existing approaches. We have evaluated the new analysis using several test cases.**

I. Introduction

A large number of communication protocols have been proposed for embedded systems. However, only a few protocols are suitable for safety-critical real-time applications [13]. The increasing number of functionality and control applications implemented on distributed real-time systems results also in an increase in the bandwidth requirements of the applications. Ethernet [7], although it is low cost and has high speeds (100 Mbps up to 10 Gbps), is known to be unsuitable for real-time and safety-critical applications [5], [8]. For example, in half-duplex implementations, frame collision is unavoidable, leading to unbounded transmission times. [5] presents the requirements for a real-time network and how Ethernet can be improved to comply with these requirements. Several real-time communication solutions based on Ethernet have been proposed, such as FTT-Ethernet [11], ARINC 664 Specification Part 7 (ARINC 664p7, for short) [2], TTEthernet [14], and IEEE Audio Video Bridging[1] (AVB). [15] and [4] describe and compare several of the proposed Ethernet-based real-time communication protocols.

In this paper we focus on the TTEthernet [14] protocol. TTEthernet is a deterministic, synchronized and congestion-free network protocol based on the IEEE 802.3 Ethernet [7] standard and compliant with the ARINC 664p7. The ARINC 664p7 specification [2] is a full-duplex Ethernet network, which emulates point-to-point connectivity over the network by defining *virtual links*, tree structures with one sender and one or several receivers (see Section II). ARINC 664p7 provides predictable event-triggered communication suitable for hard real-time applications, and separation of safety-critical messages through the concept of virtual links. In addition to the functionality offered by Ethernet and ARINC 664p7, TTEthernet supports time-triggered communication based on static communication schedules which rely on a synchronized time base. Such time-triggered static scheduling approach is especially suitable for applications with highest criticality requirements in both temporal and safety domains.

TTEthernet supports applications with mixed-criticality requirements in the temporal domain, as it provides three types of traffic: static time-triggered (TT) traffic and dynamic traffic, which is further subdivided into Rate Constrained (RC) traffic that has bounded end-to-end latencies, and Best-Effort (BE) traffic, for which no timing guarantees are provided. TT messages are transmitted based on static schedule tables and have the highest priority. RC messages are transmitted if there are no TT messages, and BE traffic has the lowest priority. TTEthernet is suitable for automotive [16], avionics [19] and space [6] applications.

The paper is organized as follows: Section I-A presents the related work. Section II and III introduce the architecture and application models. Section IV presents our proposed timing analysis for the TTEthernet protocol in detail. Section V explain the proposed analysis using an example. Section VI evaluates the proposed analysis, and Section VII concludes this paper and presents our future work.

A. Related work

Researchers have proposed several worst case end-to-end delay analyses for the traffic in an ARINC 664p7 network, including analyses based on Network Calculus [3], Timed Automata [1] or Trajectory Approach [10]. However, none of these analysis methods are applicable to TTEthernet, since they do not consider the impact of TT messages on the schedulability of RC messages.

There are very few proposed timing analyses for TTEthernet systems. The earliest analysis approach for RC traffic [17] considers that the TT schedules contain periodically alternating phases for TT traffic and for RC traffic. However, these are simplified assumptions, as realistic schedules do not contains such periodic phases. More recently, Zhao et al [22] have proposed a network calculus-based analysis to compute the

[1] Audio Video Bridging is a collection of technical specifications that target synchronized communication with low jitter and low latency on Ethernet networks.

WCD of RC frames. Both analyses [17], [22] compute the WCD of a frame by summing the worst-case delay of the frame on each dataflow link traversed by the frame, thus potentially leading to unreachable scenarios, where the worst-case delay on a dataflow link occurs in time after the situation leading to the worst-case delay on the following dataflow link. In this paper, we propose a new TTEthernet analysis that has reduced pessimism compared to the previous analyses by eliminating these unreachable scenarios and computing for each time instance in the schedule the end-to-end delay of the frame.

II. Architecture Model

TTEthernet networks are composed of one or several clusters interconnected by gateways. Each cluster groups End Systems (ESes) and Network Switches (NSes) with the same synchronization priority. The ESes and NSes are interconnected by full duplex links, allowing simultaneous communication in both directions. An example cluster is presented in Fig. 1, where we have 4 ESes, ES_1 to ES_4, and 3 NSes, NS_1 to NS_3.

We model a TTEthernet cluster as an undirected graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, where $\mathcal{V}=\mathcal{ES}\cup\mathcal{NS}$ is the set of end systems ($\mathcal{ES}$) and network switches ($\mathcal{NS}$) and $\mathcal{E}$ is the set of physical links. For Fig. 1, $\mathcal{V}=\mathcal{ES}\cup\mathcal{NS}=\{ES_1, ES_2, ES_3, ES_4\}\cup\{NS_1, NS_2, NS_3\}$, and the physical links $\mathcal{E}$ are depicted with thick, black, double arrows.

The space partitioning between messages of different criticality transmitted over physical links and network switches is achieved through the concept of *virtual link*. Virtual links are defined by ARINC 664p7 [2], which is implemented by the TTEthernet protocol, as a "logical unidirectional connection from one source end system to one or more destination end systems".

We denote the set of virtual links in a cluster with $\mathcal{VL}$. A virtual link $vl_i \in \mathcal{VL}$ is a directed tree, with the sender as the root and the receivers as leafs. Each virtual link is composed of a set of dataflow paths, one such dataflow path for each root-leaf connection. A *dataflow path* $dp_i \in \mathcal{DP}$ is an ordered sequence of dataflow links connecting one sender to one receiver. A *dataflow link* $l_i=[\nu_j,\nu_k]\in\mathcal{L}$, where $\mathcal{L}$ is the set of dataflow links in a cluster, is a directed communication connection from ν_j to ν_k, where ν_j and $\nu_k \in \mathcal{V}$ can be ESes or NSes. More formally, we denote with $\mathcal{R}_{VL}(vl_i)=\{\forall dp_j \in \mathcal{DP} | dp_j \in vl_i\}$ the routing of virtual link vl_i.

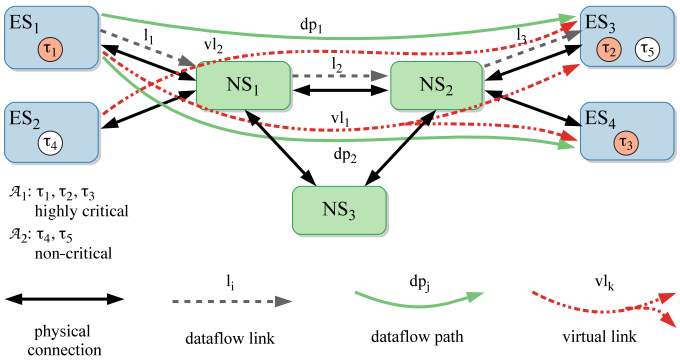


Fig. 1: TTEthernet cluster example

Let us assume that in Fig. 1 we have two applications, $\mathcal{A}_1$ and $\mathcal{A}_2$. $\mathcal{A}_1$ is a high criticality application consisting of tasks τ_1 to τ_3 mapped on ES_1, ES_3 and ES_4, respectively. $\mathcal{A}_2$ is a non-critical application, with tasks τ_4 and τ_5 mapped on ES_2 and ES_3, respectively. τ_1 sends message m_1 to τ_2 and τ_3. Task τ_4 sends message m_2 to τ_5. With TTEthernet, a message has a single sender and may have multiple receivers. The flow of these messages will intersect in the physical links and switches. Virtual links are used to separate the highly critical message m_1 from the non-critical message m_2. Thus, m_1 is transmitted over virtual link vl_1, which is isolated from virtual link vl_2, on which m_2 is sent, through protocol-level temporal and spatial mechanisms (which are presented in details in [20]).

For example, vl_1, depicted in Fig. 1 using dot-dash red arrows, is a tree with the root ES_1 and the leafs ES_3 and ES_4, and $\mathcal{R}_{VL}(vl_1)=\{dp_1,dp_2\}$. Dataflow path dp_1 connects ES_1 to ES_3, while dp_2 connects ES_1 to ES_4 (the dataflow paths are depicted with green arrows). Moreover, dp_1 can be denoted as $[[ES_1, NS_1], [NS_1, NS_2], [NS_2, ES_3]]$.

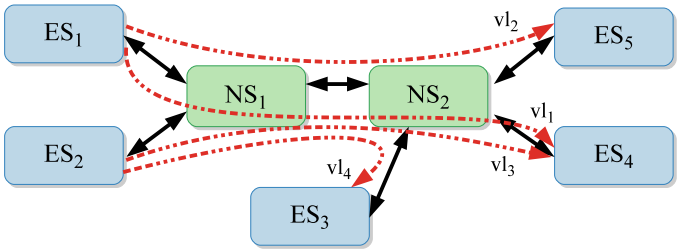
III. Application Model

TTEthernet transmits data using *frames*. The TTEthernet frame format fully complies with the ARINC 664p7 specification [2]. Messages are transmitted in the payload of frames. In [21] we proposed a design optimization of TTEthernet systems, were our assumption was that a frame can transmit several messages and that a message can be fragmented into several pieces, each carried by a different frame. However, in this work, we assume that messages are not split into packets, and that each frame transmits a single message.

The size $m_i.size$ for each message $m_i \in \mathcal{M}$ is given, where $\mathcal{M}$ is the set of all messages. As mentioned, TTEthernet supports three traffic classes: TT, RC and BE. We assume that the designer has decided the traffic classes for each message. We define the sets $\mathcal{M}^{TT}$, $\mathcal{M}^{RC}$ and $\mathcal{M}^{BE}$, respectively, with $\mathcal{M}=\mathcal{M}^{TT}\cup\mathcal{M}^{RC}\cup\mathcal{M}^{BE}$. Similarly, we define $\mathcal{F}^{TT}$, $\mathcal{F}^{RC}$ and $\mathcal{F}^{BE}$, with $\mathcal{F}=\mathcal{F}^{TT}\cup\mathcal{F}^{RC}\cup\mathcal{F}^{BE}$ is the set of all the frames in the networks. Knowing the size $m_i.size$ for each message m_i, we can compute the size $f_i.size$ of the frame f_i packing m_i. In addition, for the TT and RC frames we know their periods / rate and deadlines, $f_i.period$ or $f_i.rate$, and $f_i.deadline$, respectively.

TT frames are transmitted according to offline computed schedules. The complete set of local schedules in a cluster are denoted by $\mathcal{S}$, and the TT schedule for a dataflow link dl_j is denoted by $S_j \in \mathcal{S}$. We have presented in [20], [21] several approaches on how to obtain the schedules $\mathcal{S}$ for a cluster. In this paper, we assume the schedules $\mathcal{S}$ are given.

RC frames are not necessarily periodic, but have a minimum inter-arrival time. We define the rate of an RC frame f_i as $f_i.rate=1/f_i.period$. For each virtual link vl_i carrying an RC frame f_i the designer decides the Bandwidth Allocation Gap (BAG). A BAG is the minimum time interval between two consecutive instances of an RC frame f_i and has a value of 2^i ms, i=0..7. The BAG is set in such a way to guarantee that there is enough bandwidth allocated for the transmission of a frame on a virtual link, with $BAG_i \leq 1/f_i.rate$. The BAG is enforced by the sending ES. Thus, an ES will ensure that each BAG_i interval will contain at most one instance of f_i. The



(a) Example architecture model

Frame	period (ms)	deadline (ms)	size (B)	C_i (ms)	Source	Dest
$f_1 \in \mathcal{F}^{TT}$	32	32	683	3	ES_1	ES_4
$f_2 \in \mathcal{F}^{TT}$	32	32	555	2.5	ES_1	ES_5
$f_3 \in \mathcal{F}^{TT}$	32	32	808	3.5	ES_2	ES_3
$f_{10} \in \mathcal{F}^{RC}$	32	32	308	1.5	ES_1	ES_5
$f_{11} \in \mathcal{F}^{RC}$	32	32	555	2.5	ES_1	ES_5
$f_{12} \in \mathcal{F}^{RC}$	32	32	433	2	ES_2	ES_4
$f_{13} \in \mathcal{F}^{RC}$	32	32	433	2	ES_1	ES_4
$f_{14} \in \mathcal{F}^{RC}$	32	32	308	1.5	ES_2	ES_3

(b) Example application model

Fig. 2: Example system model

maximum bandwidth used by a virtual link vl_i transmitting an RC frame f_i is $BW(vl_i) = f_i.size/BAG(vl_i)$. The BAG for each RC frame is computed offline, based on the requirements of the messages it packs.

The routing of virtual links and the assignment of frames to virtual links are given. This assignment is captured by the function $\mathcal{M}_F : \mathcal{F} \rightarrow \mathcal{VL}$, $\mathcal{M}_F(f_i) = vl_i$. We define the set $\mathcal{F}_j^{RC} \subset \mathcal{F}^{RC}$ as the set of RC frames that are routed via dataflow link dl_j. More formally, $\mathcal{F}_j^{RC} = \{f_i \in \mathcal{F}^{RC} | dl_j \in \mathcal{M}_F(f_i)\}$. Knowing the size of a frame f_i and the given speed of a dataflow link $[\nu_m, \nu_n]$, we can determine the transmission duration $C_i^{[\nu_m,\nu_n]}$ of f_j on $[\nu_m, \nu_n]$.

IV. Timing Analysis

In this paper we are interested in determining the worst-case end-to-end delay (WCD) R_x of a frame f_x going from one ES to several other ESes via dataflow links and NSes. We assume that the cluster contains both TT and RC traffic, that the complete set of TT schedules $\mathcal{S}$ is given, and that all the RC frames have the same priority. Once the WCD R_x of frame f_x is computed, we can determine if the frame is schedulable or not, by comparing R_x with the deadline. For TT frames, the WCD is easily obtained by checking the TT schedules, which are fixed. In this paper, we focus on determining the WCD for RC frames.

We base our analysis on the concept of "busy period" [9]. For the analyzed frame f_x, the *busy period* on a dataflow link is a time interval starting when the frame arrives at the incoming network node, and ending when the frame was transmitted on the dataflow link to the next network node. During this interval, the frame can be delayed by several factors, presented in Section IV-A. The definition of the busy period is formalized in Section IV-B and we show in Section IV-C how to compute the length of the busy period on a dataflow link. The WCD R_x of frame f_x is the largest difference between the end of the busy period on the last dataflow link and the beginning of the busy period on the first dataflow link for each dataflow path f_x is transmitted on. The algorithm to compute the WCD is presented in Section IV-D. Next, we will present the source of delays for RC frames.

A. Sources of delay for RC traffic

There are several factors contributing to the delay of an RC frame since its arrival at a network node and until its transmission on the following dataflow link. The factors are: (a) technical latencies introduced by the network nodes, (b) other RC frames that are already waiting for transmission on the egress link, (c) scheduled TT frames, and (d) further delays due to the integration of TT and RC traffic. Next, we will discuss each factor in detail.

(a) **Technical latencies introduced by the network nodes**. The TTEthernet network nodes implement the protocol functionality as hardware tasks. For example, network node has a hardware task that checks the validity and integrity of each arriving frame. Hardware tasks in the NSes also provide fault-containment at the level of RC virtual links, by implementing an algorithm known as *leaky bucket* [2], [14]. The algorithm checks the time interval between two consecutive instances on the same virtual link. If this interval is shorter than the specified BAG time corrected by the maximum allowed transmission jitter, the frame instance is dropped, thus preventing a faulty ES to send faulty RC frames (more often than allowed) and disturbing the network. The TTEthernet protocol and the hardware tasks are presented in more detail in [20]. The technical latencies introduced by the network nodes implementing TTEthernet functionality increase the delay of an RC frame in a network node. We assume that these delays are not frame-specific, and are known.

(b) **Delays from other RC frames**. The TTEthernet NS contains an RC outgoing queue for each egress dataflow link. Every valid frame arriving at the NS is copied to the outgoing queues corresponding to the dataflow links the frame will be transmitted on next. RC traffic is unsynchronized, leading to situations where several RC frames are present in an outgoing queue waiting for transmission. In this paper, we assume that all the RC frames have the same priority, similar to related work, and the frames are ordered in a FIFO-manner. Thus, the time necessary to transmit the RC frames placed in the outgoing queue before the RC frame under analysis increase its delay.

(c) **Scheduled TT frames**. RC traffic also has to be integrated with TT traffic, which has higher priority. TT communication is done according to static communication schedules determined offline and stored in the ESes and NSes. RC frames are transmitted only when there is no TT traffic on the dataflow link. Therefore, the time necessary to transmit the scheduled TT frames further increases the delay of an RC frame ready for transmission.

(d) **TT and RC traffic integration-induced delays**. However, with the integration of TT and RC traffic, contention situations can occur when a TT frame is scheduled for transmission, but an RC frame is already transmitting. There are three approaches in to handle such situations [14], [18]: (i) shuffling, (ii) pre-emption and (iii) timely block. (i) With

shuffling, the higher priority TT frame is delayed until the RC frame finishes the transmission. Thus, in the worst-case scenario, the TT frame will have to wait for the time needed to transmit the largest Ethernet frame, which is 1542 Bytes. In the case (ii) of *pre-emption*, the current transmission of the RC frame is aborted, and is restarted after the TT frame finished transmitting. In the case (iii) of *timely block*, the RC frame is blocked (postponed) from transmission on a dataflow link if a TT frame is scheduled to be sent before the RC frame would complete its transmission. Among the three policies, only the "pre-emption" and "timely block" can increase the delay for RC frames. In this paper we consider the cluster implements the "timely block" integration policy. In the worst-case, this policy can delay the RC frames by, at most, the time necessary to transmit the largest TT frame on the given dataflow link.

B. Busy period on a dataflow link

We formally define the *busy period* bp_x^j on dataflow link dl_j as the time interval $[t_c^j, t^j]$ within which the RC frames arrived before f_x in the outgoing queue for dl_j are transmitted. The busy period begins at the time instant t_c^j when the analyzed frame f_x arrives in the network node outgoing queue, and ends at time t_j when all the RC frames that arrived at the queue before f_x including f_x have finished transmitting on dl_j. Since we are interested in the worst-case scenario, we assume that all the RC frames transmitted on dl_j (other than f_x) arrive at the queue before f_x. We denote by $bp_x^j(t_c^j) = t^j$ the end time of the busy period, and by $\overline{bp_x^j} = t^j - t_c^j$ the length of the busy period bp_x^j.

Next, we will explain the busy period concept using the example shown in Fig. 3. We will compute the end time t^j of the busy period bp_x^j for the RC frame f_x arriving in the outgoing queue at time instant t_c^j. Fig. 3 presents using a Gantt chart a fragment of the TT schedule on dataflow link dl_j during the time interval marked by the busy period bp_x^j starting at time instant t_c^j. The schedule fragment contains 3 TT frames f_1, f_2 and f_3 marked with red colored boxes. Besides the TT frames and the analyzed RC frame f_x, dl_j also transmits 2 other RC frames f_4 and f_5 marked with purple colored boxes.

To compute the worst-case ending time of the busy period, we have to take into account all the potential sources of delay presented in Section IV-A. Thus, the formula to compute the

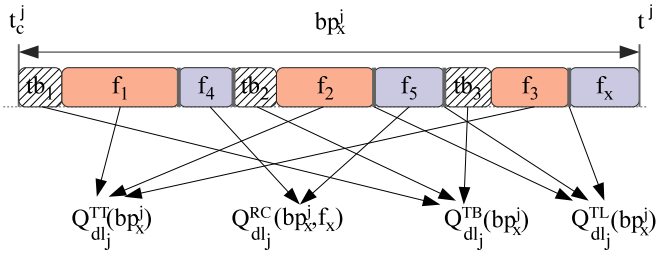


Fig. 3: Busy period starting at time instant t_c

end of the busy period bp_x^j shown in Fig. 3 is:

$$bp_x^j(t_c^j) = Q_{dl_j}^{TT}(bp_x^j) + Q_{dl_j}^{RC}(bp_x^j, f_x) + Q_{dl_j}^{TL}(bp_x^j) + Q_{dl_j}^{TB}(bp_x^j) + C_x^j \quad (1)$$

Let us now explain each term. $Q_{dl_j}^{TT}(bp_x^j)$ is the time during the busy period bp_x^j reserved in the static schedule S_j for transmission of the TT frames on dl_j, and is captured by Eq. 2:

$$Q_{dl_j}^{TT}(bp_x^j) = \sum_{\substack{f_i \in \mathcal{F}^{TT} \wedge \\ f_i \in S_j(bp_x^j)}} C_i^j \quad (2)$$

Thus, Eq. 2 sums up the transmission time C_i^j for all the TT frames f_i that are scheduled for transmission on dl_j during the busy period. We denote by $S_j([t_a, t_b])$ as the schedule for dl_j during the time interval $[t_a, t_b]$, and with $f_i \in S_j(bp_x^j)$ the frames scheduled on dl_j during bp_x^j. In the case of Fig. 3, $Q_{dl_j}^{TT}(bp_x^j) = C_1^j + C_2^j + C_3^j$.

$Q_{dl_j}^{RC}(bp_x^j, f_x)$ is the delay incurred by the RC frames arrived at the outgoing queue before f_x:

$$Q_{dl_j}^{RC}(bp^j, f_x) = \sum_{\substack{f_i \in \mathcal{F}_j^{RC} \wedge \\ f_i \neq f_x}} C_i^j \cdot \left\lceil \frac{\overline{bp_x^j}}{f_i.rate} \right\rceil \quad (3)$$

Since we are interested in the worst-case scenario for the busy period, we assume that all the RC frames transmitted on the dl_j (except f_x) will be placed in the queue before f_x. Thus, Eq. 3 captures the sum of the transmission times for these RC frames in the outgoing queue. In some cases, the length $\overline{bp_x^j}$ of the busy period might be larger than the rate $f_i.rate$ of an RC frame f_i, thus, the outgoing queue might contain several instances of f_i. The number of these instances is captured by $\left\lceil \frac{\overline{bp_x^j}}{f_i.rate} \right\rceil$. In the example shown in Fig. 3, we assume that the busy period is shorter than the rate of f_4 and f_5, and these two RC frames are already waiting in the outgoing queue when f_x arrives, therefore, they are transmitted before f_x on dl_j. Thus, $Q_{dl_j}^{RC}(bp^j, f_x) = C_4^j + C_5^j$.

In Section IV-A we also presented delays due to the implemented integration policy and due to the hardware tasks implementing the TTEthernet protocol functionality, other than the latency resulting from queueing effects. $Q_{dl_j}^{TL}(bp^j)$ captures this technical latency introduced by the network node for each frame. Moreover, in this paper we assume that the protocol implements the timely block integration policy (see Section IV-A), but it can be easily extend to consider other policies as well. In this case however, $Q_{dl_j}^{TB}(bp_x^j)$ is the sum of the timely block intervals in the busy period bp_x^j. The span of a timely block interval tb_i before a TT frame f_i is equal to the time C_k^j necessary to transmit the largest RC frame f_k on dl_j, or with the time between frame f_i and the previously scheduled TT frame, whichever is smaller. Fig. 3 contains 3 such timely block intervals (tb_1, tb_2 and tb_3), one for each of the TT frame in the schedule fragment (f_1, f_2 and f_3).

$Q_{dl_j}^{TL}(bp^j)$ is the sum of the technical latency introduced by the network node for each frame, due to the hardware tasks

1: **function** COMPUTEBUSYPERIOD(dl_j, t_c^j, $\mathcal{F}$, S_j, f_x)
2: $bp_x^j(t_c^j) = t_c^j + \sum_{f_i \in \mathcal{F}_j^{RC}} C_i^{dl_j}$
3: **repeat**
4: compute demand $H_x^j(bp_x^j)$
5: compute availability $A_x^j(bp_x^j)$
6: **if** $H_x^j(bp_x^j) > A_x^j(bp_x^j)$ **then**
7: $t^j = t^j + H_x^j(bp_x^j) - A_x^j(bp_x^j)$
8: **end if**
9: **until** $A_x^j(bp_x^j) \geq H_x^j(bp_x^j)$
10: **return** $\overline{bp_x^j} = t^j - t_c^j$
11: **end function**

Fig. 4: Iterative algorithm to compute the busy period

implementing the TTEthernet protocol functionality, other than the latency resulting from queueing effects.

C. Computing the length of a busy period

Eq. 1 is a recursive function, with bp_x^j on both sides of the equality. In order to compute the length $\overline{bp_x^j}$ of the busy period bp_x^j, we use the concepts of *ET availability* and *ET demand* [12], which we have adapted from tasks to messages, as follows. The *demand* for a frame f_x on a dataflow link dl_j during the busy period bp_x^j is the maximum amount of transfer time which can be demanded by RC frames arrived at the outgoing queue before f_x and by frame f_x. The *demand* $H_x^j(bp_x^j)$ during the busy period bp_x^j is equal to the length of the busy period if considering only RC traffic:

$$H_x^j(bp_x^j) = Q_{dl_j}^{RC}(bp_x^j, f_x) + C_x^j \quad (4)$$

The *availability* is the fraction of time available for RC traffic during the busy period bp_x^j. The availability A_x^j is computed by subtracting from the length $\overline{bp_x^j}$ of the busy period the time necessary for the transmission of TT frames, reserved by the timely block intervals, or by the technical latency:

$$A_x^j(bp_x^j) = \overline{bp_x^j} - (Q_{dl_j}^{TT}(bp_x^j) + Q_{dl_j}^{TB}(bp_x^j) + Q_{dl_j}^{TL}(bp_x^j)) \quad (5)$$

We compute the length of the busy period using the algorithm presented in Fig. 4. The function takes as input the dataflow link dl_j, the critical instant t_c^j starting the busy period bp_x^j, the complete set of frames $\mathcal{F}$, the TT schedules S_j for dl_j, and the frame to be analyzed f_x. We initialize the busy period bp_x^j starting at t_c^j with a length equal to the sum of all the RC frames transmitted on dl_j, ignoring any intervals unavailable due to the transmission of TT frames (line 2 in Fig. 4). Next, we iteratively compute the length of the busy period bp_x^j until the demand H_x^j (see Eq. 4) is satisfied by the availability A_x^j (see Eq. 5) during the busy period (lines 3–9). The availability satisfies the demand when there is enough available time during the busy period to transmit all the RC frames in the outgoing queue. First, we update the value of the demand (line 4) and of the availability (line 5) for the new interval of the busy period bp_x^j. If A_x^j does not satisfy H_x^j, we extend the busy period with the difference between the demand and the availability (line 7). Once the demand is satisfied, the algorithm returns the length $\overline{bp_x^j}$ of the busy period bp_x^j starting at t_c^j. Section V presents an example on how the length of the busy period is computed.

1: **function** COMPUTEDELAY(t_c^0, $\mathcal{F}$, $\mathcal{S}$, f_x, vl_x)
2: Delay $= 0$
3: **for** $dp_i \in R_{VL}(vl_x)$ **do**
4: **for** $dl_j \in dp_i$ **do**
5: compute t_c^j based on bp_x^{j-1}
6: $t^j = t_c^j + ComputeBusyPeriod(dl_j, t_c^j, \mathcal{F}, S_j, f_x)$
7: **end for**
8: **if** $t^n - t_c^0 > R_x$ **then**
9: Delay = $t^n - t_c^0$
10: **end if**
11: **end for**
12: **return** Delay
13: **end function**

Fig. 5: Algorithm to compute the end-to-end delay for a frame

D. Worst-case end-to-end delay

All the related work compute the WCD R_i of an RC frame f_i by summing up the worst-case delays on each dataflow link traversed by f_i. However, this might lead to unlikely cases, such as, the TT schedule in the interval leading to the worst-case busy period on a dataflow link dl_j might occur in time after the TT schedule in the interval leading to the worst-case busy period on the following dataflow link dl_{j+1}. To obtain tight WCDs (i.e., very close to the actual worst-case), our analysis attempts to eliminate these scenarios which cannot occur. Fig. 5 presents the algorithm to compute the WCD of frame f_x on virtual link vl_x, starting at time instance t_c^0. The algorithm receives as input also the set of frames $\mathcal{F}$ and the set of schedules in the system $\mathcal{S}$.

The algorithm computes the end-to-end delay by calculating the longest delay of dataflow path dp_i in virtual link vl_x (lines 3–11 in Fig. 5). In the case of unicast frames, i.e., frames with only one destination ES, the virtual link contains only one dataflow path. The delay on a path is computed as the time difference between the end t^n of the busy period bp_x^n on the last dataflow link dl_n in the dataflow path and the critical instant t_c^0 starting the busy period on the first dataflow link in the current dataflow path. For each dataflow link $dl_j \in dp_i$, we compute the start time t_c^j of the busy period bp_x^j based on the start time t_c^{j-1} of the busy period on the previous dataflow link dl_{j-1} (line 5). We obtain t_c^j by subtracting from the end t^{j-1} of the busy period on dl_{j-1} the time necessary to transmit all the RC frames $f_k \in \mathcal{F}_{j-1}^{RC} \cap \mathcal{F}_j^{RC}$, i.e., all the RC frames transmitted during the busy period bp^{j-1} on dl_{j-1} that are also transmitted on dl_j. In the case of the first dataflow link in dp_i, the critical instant starting the busy period is equal to t_c^0.

The WCD R_x of frame f_x is obtained by computing the delay for each time instance t_c^0 in the schedule S_0 of the first dataflow link dl_0 transmitting frame f_x, and selecting the time instance resulting in the biggest delay.

V. TIMING ANALYSIS EXAMPLE

In this section we will show how the timing analysis presented in the previous section computes the WCD R_x of

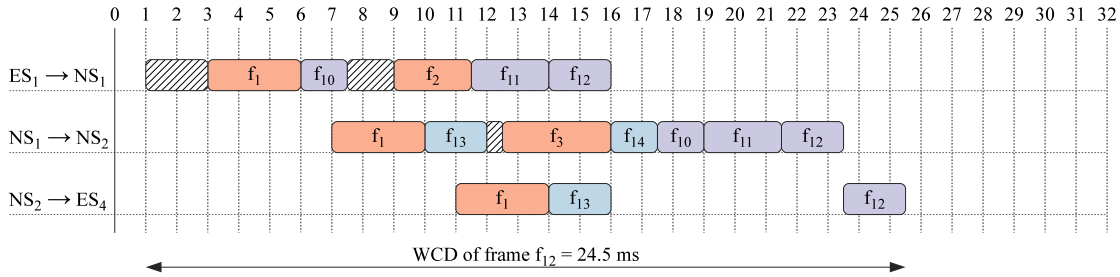


Fig. 6: Exact worst-case end-to-end delay for f_{12}

a frame f_x. Let us present this analysis using the application details presented in Fig. 2. In Fig. 2 we have a cluster composed of 5 end systems, ES_1 to ES_5 and two network switches, NS_1 and NS_2. The table in Fig. 6 presents the frames in the network: we have 3 TT frames (f_1 to f_3), and 5 RC frames (f_{10} to f_{14}). The frame details (the period, the deadline, the payload size, and the transmission time) are given in the table. Although the standard TTEthernet speed is 100 Mbps or higher, for the sake of this example we consider a link speed of only 2 Mbps, and that all the dataflow links have the same speed. Furthermore, for simplicity, we consider one destination ES for each message and the technical latency is zero.

We will consider for this example the end-to-end delay of frame f_{12}. We assume the TT schedules presented in Fig. 6 as a Gantt chart. Since we are interested in the WCD for this frame, we show in Fig. 6 only the schedules for the dataflow links transited by f_{12}. We have marked with boxes with hatching pattern the timely block intervals (see Section III for a brief description of timely block). Fig. 6 also presents the exact WCD of frame f_{12}, which is $R_{12} = 24.5$ms.

First, we will compute the worst-case end-to-end delay using the analysis presented in [17], to highlight the pessimism of the only TTEthernet analysis available now. The analysis in [17] computes the WCD of a frame by summing up the worst-case delay of the frame on each dataflow link the frame is transmitted on. This analysis assumes that the schedule has periodically alternating phases for TT and RC frames. The worst-case delay of a frame on a dataflow link is computed by taking into account the time necessary to transmit all the RC frames in the outgoing queueing (considering there are no TT frames), and the number of RC phases necessary to transmit the RC frames.

The authors introduce the term $\widehat{burst}^{[\nu_x,\nu_k]}_{[\nu_k,\nu_y]}$ as the necessary time to transmit the RC frames received from dataflow link $[\nu_x,\nu_k]$ and forwarded to $[\nu_k,\nu_y]$, and $\widehat{BURST}^{in}_{[\nu_k,\nu_y]}$ as the total time necessary to transmit all the RC frames on $[\nu_k,\nu_y]$. Considering the example in Fig. 6, with the application details presented in Fig. 2, $\widehat{burst}^{[ES_1,NS_1]}_{[NS_1,NS_2]} = C^{[ES_1,NS_1]}_{10} + C^{[ES_1,NS_1]}_{11} + C^{[ES_1,NS_1]}_{12}$, while $\widehat{burst}^{[ES_2,NS_1]}_{[NS_1,NS_2]} = C^{[ES_2,NS_1]}_{13} + C^{[ES_2,NS_1]}_{14}$.

Furthermore, the analysis in [17] defines l_{TT} as length of the TT phase, and l_{blank} as the length of the RC phase. However, realistic schedules do not have such phases, as this is just a simplifying assumption of the analysis in [17]. To map a realistic schedule on the analysis from [17], we need to redefine l_{TT} as the maximum time necessary to transmit consecutive TT frames, while l_{blank} is redefined as the minimum time between two TT frames.

Based on these values, [17] defines the maximum time necessary to transmit all the RC frames in the outgoing queue as:

$$\widehat{Q}^{out}_{[\nu_k,\nu_y]} = \widehat{BURST}^{in}_{[\nu_k,\nu_y]} - max(\widehat{burst}^{[\nu_x,\nu_k]}_{[\nu_k,\nu_y]}) + l_{TT} \times \left(\left\lfloor \frac{max(\widehat{burst}^{[\nu_x,\nu_k]}_{[\nu_k,\nu_y]})}{l_{TT} + l_{blank}} \right\rfloor + 1 \right) \quad (6)$$

and the worst-case delay for the RC frame f_i is given by the equation below:

$$\widehat{latency}^{[\nu_x,\nu_k]}_{[\nu_k,\nu_y]} = \widehat{Q}^{out}_{[\nu_k,\nu_y]} + C_i^{[\nu_k,\nu_y]} + \left\lceil \frac{\widehat{Q}^{out}_{[\nu_k,\nu_y]}}{l_{blank}} \right\rceil \times l_{TT} \quad (7)$$

where the first right term is the time necessary to transmit the RC frames, the second term is the time necessary to transmit the frame under analysis, and the third term captures the number of TT phases interrupting the transmission of the RC frames.

Thus, using the analysis in [17], the WCD for f_{12} is computed by summing the worst-case latency computed using Eq. 7 on all the dataflow links traversed by f_{12}. On [ES_1, NS_1], $l_{TT} = 3$ (the size of f_1) and $l_{blank} = 3$. Using Eq. 6, we compute $\widehat{Q}^{out}_{[ES_1,NS_1]} = 6$, resulting in a worst-case latency for f_{12} of 14. On [NS_1, NS_2], with $l_{TT} = 3.5$ and $l_{blank} = 2.5$, $\widehat{Q}^{out}_{[NS_1,NS_2]} = 8$ and the worst-case latency is 24. On the last dataflow link, $l_{TT} = 3$ and $l_{blank} = 37$, resulting in a worst-case latency of 9. Adding up the latencies on all the dataflow links results in a WCD for f_{12} of 47, almost double the value of the exact WCD, which is 24.5 ms. The pessimism in the computed WCD is mainly due to the fact that the analysis considers that the schedules have alternating phases for TT and RC frames, thus greatly underestimating the actual available time for transmitting RC frames. For example, on dataflow link [NS_1, NS_2] the analysis considers that the schedule has a TT phase of 3.5 ms and an RC phase of 2.5 ms.

Fig. 7 shows how our analysis computes the WCD for frame f_{12}. We will present this analysis step by step, considering that the analysis starts from the same critical instant $t_c = 1$ that led to the worst-case scenario presented in Fig. 6. We obtain the time-critical leading to the WCD by computing the end-to-end delay using the algorithm presented in Fig. 5 for each time instant in the schedule $S_{[ES_1,NS_1]}$. Let us first compute the busy period on [ES_1, NS_1], using the algorithm presented in Fig. 4. First we initialize the length of the busy period

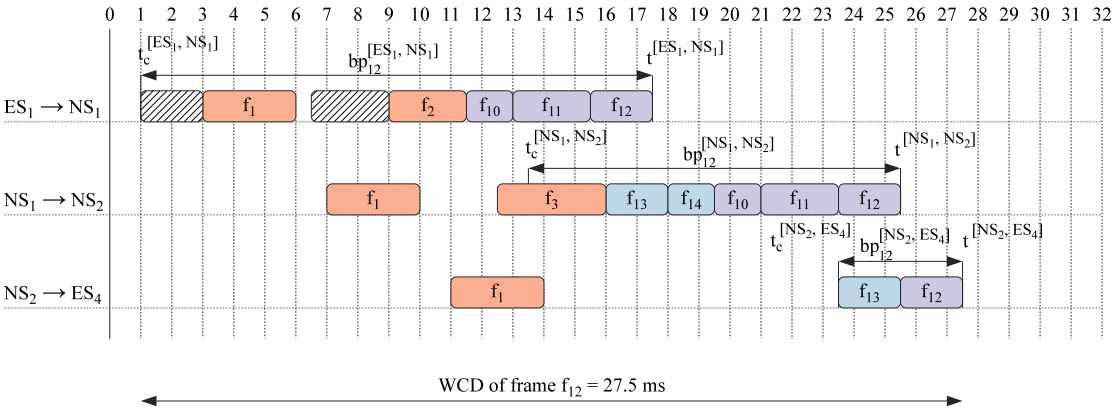


Fig. 7: Analysis example

with the sum of the necessary time to transmit the RC frames on $[ES_1, NS_1]$, i.e., f_{10} to f_{12}. Thus, $bp_{12}^{[ES_1,NS_1]}$ is initialized with the interval [1, 7]. The demand $H_{12}^{[ES_1,NS_1]} = 6$, i.e., the sum of the RC frames. The availability $A_{12}^{[ES_1,NS_1]} = 0$: although there is available time in the interval [6, 6.5], it is followed by a timely block interval, and it is too small to transmit even the smallest frame, so it is ignored. Thus, the busy period is incremented with $H_{12}^{[ES_1,NS_1]} - A_{12}^{[ES_1,NS_1]} = 6$. The busy period is updated to [1, 13]. In this interval, $H_{12}^{[ES_1,NS_1]}$ has the same value, while $A_{12}^{[ES_1,NS_1]} = 1.5$. Although the interval is smaller than the smallest RC frame transmitted on this link, the busy period is not followed by a timely block interval. Therefore, there is potentially enough time to transmit a frame afterwards. Again, we increase the busy period interval with the difference between the demand and availability, i.e., 4.5. The new busy period is [1, 17.5]. In the new interval, the availability satisfies the demand, therefore this is the final interval for $bp_{12}^{[ES_1,NS_1]}$.

Next, we compute the starting time $t^{[NS_1,NS_2]}$ of the busy period $bp_{12}^{[NS_1,NS_2]}$, by subtracting from the end time $t^{[ES_1,NS_1]}$ of the busy period $bp_{12}^{[ES_1,NS_1]}$ the transmission time of frames f_{10} and f_{11}. Therefore $t^{[NS_1,NS_2]} = 17.5 - 4 = 13.5$. Afterwards, we compute the busy period $bp_{12}^{[NS_1,NS_2]}$ using the algorithm from Fig. 4, and we obtain the interval [13.5, 25.5]. Similarly, we obtain $bp_{12}^{[NS_2,ES_4]}$ as the time interval [25.5, 27.5].

The end-to-end delay of frame f_{12} computed with our proposed analysis is $t^{[NS_2,ES_4]} - t_c^{[ES_1,NS_1]} = 26.5$. Compared to the analysis from [17], our proposed analysis reduces considerably the pessimism (26.5 ms compared to 47 ms). However, compared to the exact WCD (shown in Fig. 6), the proposed analysis is a bit more conservative due to how it reserves the timely block intervals. Our proposed analysis, reserves a timely block interval equal to the time necessary to transmit the largest RC frame on the dataflow link, before each TT frame, regardless (a) if the largest RC frame is the frame under analysis, and regardless (b) of the RC frame sizes. However, not much can be done to reduce this pessimism.

VI. Evaluation

We have evaluated the proposed analysis using three synthetic use cases. The first use case has a topology of 12 ESes and 4 NSes (shown in Fig. 8), running 20 TT frames and 26 RC frames. The details of the frames and the results of the first test case are presented in Table I. The frame name, the size and period are presented in columns 1, 2 and 3, respectively. Column 4 presents the WCDs computed with the analysis from [17], while column 5 presents the results obtained with our proposed analysis. The last column shows the difference between the results obtained with the two analyses. As we can see, our proposed analysis reduces the pessimism by an order of magnitude compared to the analysis from [17]. However, while the analysis from [17] requires only 0.415 s to compute the WCDs, our analysis takes 6 minutes.

The following two use cases have increasing complexity. The second use case has 58 TT messages and 51 RC messages transmitted on a cluster composed of 10 ESes and 5 NSes. The sizes of the RC frames range between 1000 and 1470 B. In this use case, our proposed analysis reduces the pessimism of the analysis from [17] with an average of 236.47 ms, with a minimum of 15 ms and maximum 412.87 ms. However, since our analysis computes the WCD by analyzing the delay at each time instant, it takes 164 minutes to compute the WCDs for this use case, compared to 0.617 s, the time necessary for the analysis from [17]. The time necessary to compute the WCD for a set of RC frames depends on both the number of frames in the set, and the size of the schedule.

The third use case is the largest, with 91 TT messages and 81 RC messages implemented on a network composed of 35 ESes interconnected by 8 NSes. In this case, our proposed

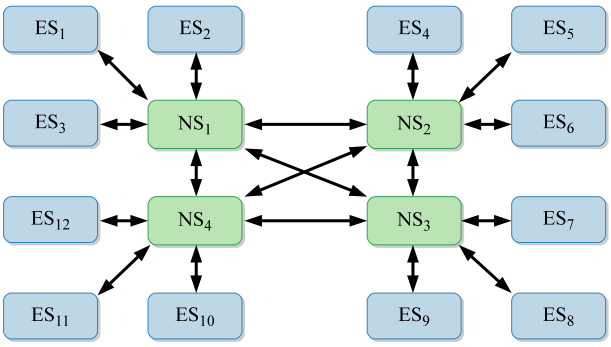


Fig. 8: Topology of the first use case

TABLE I: Experimental results, first use case

Frame	Size (B)	Period (ms)	WCD using [17] (ms)	Our WCD (ms)	Difference (ms)
rc_1	1021	4	4.44	0.77	3.66
rc_2	1395	16	19.94	1.81	18.12
rc_3	134	4	20.68	1.10	19.57
rc_4	1078	2	10.16	1.53	8.62
rc_5	590	8	13.04	1.35	11.69
rc_6	946	2	14.62	1.68	12.93
rc_7	784	16	3.12	0.79	2.33
rc_8	1120	2	14.09	1.22	12.86
rc_9	1361	8	8.43	1.38	7.04
rc_{10}	20	4	17.81	1.48	16.33
rc_{11}	1262	8	11.30	1.34	9.96
rc_{12}	926	4	15.30	1.17	14.13
rc_{13}	879	4	12.86	1.43	11.43
rc_{14}	1360	16	16.69	1.80	14.89
rc_{15}	1332	8	14.62	1.60	13.01
rc_{16}	728	16	13.67	1.61	12.05
rc_{17}	1127	16	18.52	1.70	16.81
rc_{18}	156	4	5.57	0.86	4.71
rc_{19}	378	8	20.73	1.08	19.65
rc_{20}	1443	2	20.07	1.75	18.31
rc_{21}	1367	2	20.52	1.85	18.67
rc_{22}	519	16	13.24	1.32	11.91
rc_{23}	522	2	19.74	1.33	18.41
rc_{24}	308	16	11.15	1.23	9.91
rc_{25}	411	2	11.11	0.65	10.46
rc_{26}	406	16	7.47	1.35	6.11

analysis reduces the pessimism of the analysis from [17] with an average 263.42 ms. For this case it took 446 minutes to compute the WCDs for the 81 RC messages, compared to the analysis from [17], which required only 0.49 s.

The time necessary to compute the WCDs depends on both the number of RC frames, and on the size of the schedules (the larger the schedules, the more time instances our analysis has to check).

VII. Conclusions and Future Work

In this paper we have proposed a timing analysis for the TTEthernet protocol. Highly suitable for mixed-criticality systems, both in the temporal and safety domain, TTEthernet offers three types of traffic classes, Time-Triggered, Rate Constrained and Best Effort. In the safety domain, the protocol offers separation between mixed-criticality frames using the concept of virtual links, and protocol-level specialized dependability services.

The proposed analysis considers that each frame packs a single message, and that the network implements the "timely block" integration policy. However, the analysis can be easily extended to handle the other policies. The results on several synthetic benchmarks show that compared to previous analyses, the proposed analysis considerably reduces the pessimism and is much closer to the exact worst-case end-to-end delay.

The current analysis obtains the worst-case end-to-end delay by computing the latencies for each time instance in the schedule. In our future work, we plan to reduce the time instances considered by the analysis, thus also considerably reducing the time necessary to obtain the result.

Acknowledgment

The research leading to these results has received funding from the Advanced Research & Technology for Embedded Intelligence and Systems (ARTEMIS) Joint Undertaking within the project EMC^2, under grant agreement no. 621429.

References

[1] M. Adnan, J.-L. Scharbarg, J. Ermont, and C. Fraboul. Model for worst case delay analysis of an AFDX network using timed automata. In *Proceedings of the Conference on Emerging Technologies and Factory Automation*, pages 1 –4, 2010.

[2] ARINC. *ARINC 664P7: Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network*. ARINC (Aeronautical Radio, Inc), 2009.

[3] M. Boyer and C. Fraboul. Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus. In *Proceedings of the International Workshop on Factory Communication Systems*, pages 11–20, 2008.

[4] 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. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 5(1):72–88, 2012.

[5] J. D. Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, 93(6):1102–1117, 2005.

[6] M. Fletcher. Progression of an open architecture: from Orion to Altair and LSS. White paper S65-5000-20-0, Honeywell, International, 2009.

[7] IEEE. *IEEE 802.3 - IEEE Standard for Ethernet*. The Institute of Electrical and Electronics Engineers, Inc., 2012.

[8] Y.-H. Lee, E. Rachlin, and P. A. Scandura. Safety and Certification Approaches for Ethernet-Based Aviation Databuses. Technical Report DOT/FAA/AR-05/52, Federal Aviation Administration, December 2005.

[9] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 201–209, Dec 1990.

[10] X. Li, O. Cros, and L. George. The trajectory approach for AFDX FIFO networks revisited and corrected. In *Proceedings of the International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10, Aug 2014.

[11] P. Pedreiras, P. Gai, L. Almeida, and G. Buttazzo. FTT-Ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems. *IEEE Transactions on Industrial Informatics*, 1(3):162–172, Aug 2005.

[12] T. Pop, P. Pop, P. Eles, and Z. Peng. Analysis and optimisation of hierarchically scheduled multiprocessor embedded systems. *International Journal of Parallel Programming*, 36(1):37–67, 2008.

[13] J. Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, 2001.

[14] SAE. *AS6802: Time-Triggered Ethernet*. SAE International, 2011.

[15] S. Schneele and F. Geyer. Comparison of IEEE AVB and AFDX. In *Proceedings of the Digital Avionics Systems Conference*, pages 7A1–1–7A1–9, 2012.

[16] T. Steinbach, H.-T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz. Tomorrow's In-Car Interconnect? A Competitive Evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802). In *IEEE Vehicular Technology Conference*, pages 1–5, September 2012.

[17] W. Steiner. Synthesis of Static Communication Schedules for Mixed-Criticality Systems. In *Proceedings of the International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pages 11–18, 2011.

[18] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan. TTEthernet Dataflow Concept. In *Proceedings of the International Symposium on Network Computing and Applications*, pages 319–322, 2009.

[19] J. Suen, R. Kegley, and J. Preston. Affordable avionic networks with Gigabit Ethernet assessing the suitability of commercial components for airborne use. In *Proceedings of SoutheastCon*, pages 1–6, 2013.

[20] D. Tămaş-Selicean, P. Pop, and W. Steiner. Synthesis of Communication Schedules for TTEthernet-based Mixed-Criticality Systems. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis*, pages 473–482, 2012.

[21] D. Tămaş-Selicean, P. Pop, and W. Steiner. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems*, 2014.

[22] L. Zhao, H. Xiong, Z. Zheng, and Q. Li. Improving worst-case latency analysis for rate-constrained traffic in the time-triggered ethernet network. *IEEE Communications Letters*, 18(11):1927–1930, 2014.