Reduced Buffering Solution for Multi-Hop HaRTES Switched Ethernet Networks

Mohammad Ashjaei[1], Moris Behnam[1], Paulo Pedreiras[2], Reinder J. Bril[1,4], Luis Almeida[3], Thomas Nolte[1]
[1] MRTC/Mälardalen University, Västerås, Sweden
[2] DETI/IT/University of Aveiro, Aveiro, Portugal
[3] IT/DEEC/University of Porto, Portugal
[4] Technische Universiteit Eindhoven (TU/e), The Netherlands

***Abstract*—In the context of switched Ethernet networks, multi-hop communication is essential as the networks in industrial applications comprise a high amount of nodes, that is far beyond the capability of a single switch. In this paper, we focus on multi-hop communication using HaRTES switches. The HaRTES switch is a modified Ethernet switch that provides real-time traffic scheduling, dynamic Quality-of-Service and temporal isolation between real-time and non-real-time traffic. Herein, we propose a method, called Reduced Buffering Scheme, to conduct the traffic through multiple HaRTES switches in a multi-hop HaRTES architecture. In order to enable the new scheduling method we propose to modify the HaRTES switch structure. Moreover, we develop a response time analysis for the new method. We also compare the proposed method with a method previously proposed, called Distributed Global Scheduling, based on their traffic response times. We show that, the new method forwards all types of traffic including the highest, the medium and the lowest priority, faster than the previous method in most of the cases. Furthermore, we show that the new method performs even better for larger networks compared with the previous one.**

I. Introduction

The complexity of the communications in networked real-time embedded systems has increased over the last decades such that the conventional communication protocols have shown to be impotent. Advances in embedded equipments, increments in their functionalities and a massive amount of exchanged data within the network are the origin of this complexity. For instance, in automotive industries using innovative applications and entertainment devices is becoming popular, which requires a high bandwidth network to support them. Meanwhile, new challenging requirements have been raised in real-time communications. Such requirements include handling a combination of traffic activation types (event-triggered and time-triggered) and dynamic reconfiguration support.

Ethernet technology was introduced as one of the promising approaches for the communication among embedded systems due to its properties such as low cost, availability and expandability. Nevertheless, the non-deterministic behavior of COTS Ethernet diminished its use in time-critical applications.

Consequently, many real-time Ethernet protocols (e.g., [1] and [2]) have been developed preserving the profits of traditional Ethernet technology such as high throughput and wide availability, while providing determinism. However, it turned out that, despite the performance and timeliness guarantee of these protocols, they have expressed severe limitations when applied on dynamic real-time applications. For instance, they cannot tolerate evolving requirements such as on-line reconfiguration of components (e.g., add or remove of streams).

As a result, the Hard Real-Time Ethernet Switching architecture (HaRTES) [3] has been developed in order to deliver adaptivity, dynamicity and effective distribution of resources in networked embedded systems. HaRTES supports real-time periodic, real-time sporadic and non-real-time traffic. The former is classified as *synchronous traffic* and the second is categorized as *asynchronous traffic*. Also, HaRTES creates temporal isolation among different traffic types by reserving specific bandwidth for each. The non-real-time traffic is scheduled in the background.

The networked embedded systems in industrial applications comprise a high amount of nodes, which is far beyond the capability of a single switch. Thus, the multi-hop communication is necessary for such applications. In the work presented in [4], we proposed a scheduling method, called Distributed Global Scheduling (DGS), to forward messages through multiple HaRTES switches. However, the DGS method produces high response times due to the specification of the method which messages are buffered in all switches in their route. In this paper, we propose a new solution, which compared with the DGS method, delivers the traffic to the destination much faster in most of the cases.

The main contributions of this paper are:

1) We propose a new scheduling method, named Reduced Buffering Scheme (RBS), to forward the traffic through multiple HaRTES switches in a multi-hop HaRTES architecture.
2) We modify the HaRTES switch structure to be able to apply the new scheduling method.
3) We develop a response time analysis for both synchronous and asynchronous traffic in the RBS method. In order to perform the analysis, we use an algorithm to capture the behavior of the RBS method, which is different than the analysis in [4].
4) We compare the RBS method with the DGS method using the proposed response time analysis. We show that the response time of the messages in the RBS method is lower than in the DGS method in most of the cases.

The rest of the paper is organized as follows. The next section describes the related work. Then, Section III presents the HaRTES architecture. Section IV presents the DGS method,

while Section V proposes the RBS method. Section VI presents the system model, and Section VII describes the response time analysis. Section VIII shows the evaluation of the proposed method and finally Section IX concludes the paper and presents some directions for future work.

II. Related Work

The literature on switched Ethernet is vast and there have been many works addressing its adequacy to real-time communication. There are relatively old research proposals such as EtheReal [5] and the EDF Scheduled Switch [6], both based on channel reservations supported on enhanced switches.

Later, many solutions, which eventually made it to the market, were proposed such as TTEthernet [1] and PROFINET IRT [2], both optimized for time-triggered operation. Also, EtherCAT [7] was developed, which is optimized for quick forwarding with dynamic update of the Ethernet frames while traversing the nodes.

Moreover, AFDX [8] is developed as a network communication specification with enhanced forwarding. AFDX has been used mainly in avionics. More recently, Audio Video Bridging (AVB) [9], as a set of technical standards developed by IEEE, is gaining a momentum, and it is mainly designed for use in the automotive industry. Ethernet AVB supports clock synchronization, bandwidth reservation and traffic shaping services. Nevertheless, this protocol has some intrinsic limitations, such as the low number of priorities (maximum 8) and lack of explicit support of synchronous traffic. These limitations restricted the capabilities of the protocol in terms of specification of the stream properties.

Several solutions were also researched based on overlay protocols that control the traffic submitted to COTS switches. Ethernet POWERLINK [10] and the FTT-SE [11] protocol, both using master-slave techniques, are such those solutions.

The FTT-SE protocol provides a bandwidth-efficient solution, however it presents some structural limitations because of using COTS switches. In fact, all nodes need to be FTT-compliant. This leads to have a specific network device driver in the operating system. The HaRTES switch overcomes this problem by inserting the master module inside the switch. Thus, the HaRTES switch maintains the capabilities of adding traffic confinement. In this paper, we focus on the HaRTES switch.

Despite the similarities between FTT-SE and HaRTES, there are also subtle but important differences, which have a strong impact in the operation and performance of both protocols. In particular, in HaRTES it is possible to have different reserved bandwidth in different links, according to the actual load. Moreover, unlike the COTS switches, HaRTES has ability to buffer the traffic. Therefore, the results previously developed for multi-hop communication in the FTT-SE networks (e.g. [12]) would result in sub-optimal performance. These, in fact, are the main reasons that motivated this work.

Regarding the timing analysis of multi-hop Ethenet networks, several methods are utilized. In the work presented in [13], Network Calculus is used to analyze the end-to-end delays of the traffic in a single-master and multi-switch network topology using the FTT-SE protocol, presented in [14]. In [15] three methods are used to derive the end-to-end traffic delays in a multi-hop AFDX network. These three methods include Network Calculus, network simulation and model checking, among which Network Calculus exhibited a higher pessimism. A tighter end-to-end delay analysis for AFDX networks is achieved using the trajectory approach as presented in [16]. However, in [17], the authors showed that for some corner cases the trajectory approach introduces some optimism, even though these corner cases have not existed in any AFDX configuration.

In the context of Ethernet AVB, the work presented in [18] has utilized the Network Calculus method to derive traffic end-to-end delays. Also, the work in [19] presents a worst-case delay verification of in-vehicle Ethernet networks using the same analytical framework to generate upper bounds and checking them against experiments in worst-case scenarios.

A different approach is followed in [20] and [21] that derive end-to-end delay bounds for a single flow in FIFO multiplexed sink-tree networks using a modified Network Calculus framework. These works use partitioning of a network topology into a set of logically separated sink-trees having egress nodes at the root and ingress nodes at the leaves. The traffic is aggregated in the nodes by introducing a FIFO policy called aggregated scheduling. A class of service curves is introduced to determine the service that is received in an aggregate scheduling network. Furthermore, the work in [22] utilized the mentioned method to investigate an admission control in sink-tree networks.

In our previous work presented in [12], we computed the worst-case end-to-end delay of traffic for multi-hop communication using the FTT-SE protocol and we compared the results with the ones computed using Network Calculus. We showed that, Network Calculus generates higher pessimism.

In this paper we propose a response time analysis based on an algorithm for the Reduced Buffering Scheme method, as Network Calculus showed higher pessimism in similar cases.

III. HaRTES Architecture

In this section, we describe the functional structure of the HaRTES switch and we present the traffic scheduling in a network using the HaRTES switch.

A. HaRTES Switch Structure

The HaRTES switch is a modified Ethernet switch based on a master-slave technique where the master module is developed inside the switch. The abstract functional structure of the HaRTES switch is illustrated in Figure 1. The packets arriving at the input ports are analyzed by the Packet Classification module, which is implemented for each input port. This module distinguishes the traffic types and appends them to the associated memory queue in the Memory Pool module, i.e., the packets, depending on their types, are stored in different memory sections of the Memory Pool.

The Master module contains the scheduler, admission control, QoS management and a repository of the traffic attributes. The traffic attributes include the deadline, minimum inter-arrival time/period, message length and priority. The scheduler in the Master module will be described in Section III-B.

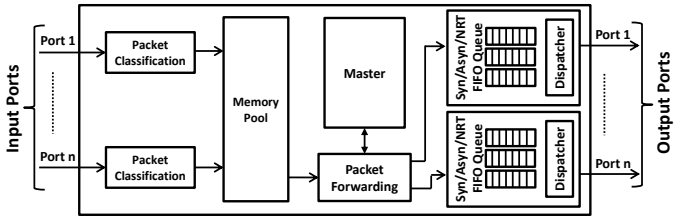

Fig. 1. HaRTES Functional Structure

For each output port three FIFO queues are implemented to handle synchronous, asynchronous and non-real-time packets, which are identified as Syn, Asyn and NRT respectively in Figure 1. The dispatcher allows packet transmission from each queue during the associated windows only, hence it handles the transmission in the reserved bandwidths, and thereby enforcing temporal isolation.

The Packet Forwarding module inquires the repository to determine the set of ports where the consumers of the packet are attached and it inserts the packet into the output FIFO queue based on the packet type. Note that, the non-real-time packets are forwarded the same way as done by standard Ethernet switches based on the MAC address. Thus, the HaRTES switch behaves as a COTS switch for the non-real-time traffic, yet with restricted bandwidth reserved for such a traffic.

B. HaRTES Traffic Scheduling

The HaRTES architecture is a micro-segmented network composed by HaRTES switches. The Master module is responsible to schedule the traffic in fixed-duration time-slots, designated Elementary Cycle (EC). The scheduling is carried out on-line according to any desired scheduling, e.g., Fixed Priority Scheduling Policy. The EC is divided between two windows, one for scheduling the synchronous traffic (Synchronous Window) and the other one for asynchronous traffic (Asynchronous Window), as shown in Figure 2. Note that, non-real-time traffic is transmitted within the asynchronous window after transmission of asynchronous traffic.

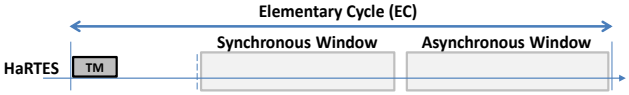

Fig. 2. The EC Partitioning in HaRTES

In each EC the switch determines the new activations of the synchronous messages, it updates the ready queue and checks whether the ready synchronous messages can be transmitted within their associated window. The scheduled messages are encoded into a particular message, named Trigger Message (TM), to be transmitted to the slave nodes at the beginning of the EC (Figure 2). The messages that do not fit in the window are kept in the ready queue for the following ECs. The slave nodes receive the TM, decode it and initiate the transmission of the messages identified in the TM.

Conversely, the activation of asynchronous messages are unknown in advance. In the HaRTES architecture, the asynchronous messages are transmitted autonomously without being triggered by the associated switch. The switch forwards them immediately through a hierarchy of servers [3]. Note that the asynchronous messages are not allowed to be transmitted within the synchronous window. Thus, during the synchronous window, such messages are buffered in the switch.

The HaRTES switch generates two types of delay, known as *store-and-forward* and *hardware fabric latency*. The former corresponds to the time required to receive the message before forwarding it, hence it is equal to the message size in time, whereas the latter delay is due to the processing speed of the switch. The hardware fabric latency is a bounded value. The summation of the two delays is called *switching delay*.

All messages that are scheduled to be transmitted in one EC, should be received by the end of the EC. In order to prevent overruns, scheduling of messages that cannot be fully transmitted within the transmission window is delayed for the next EC, e.g., m_3 in Figure 3. This behavior introduces an idle time in each transmission window.

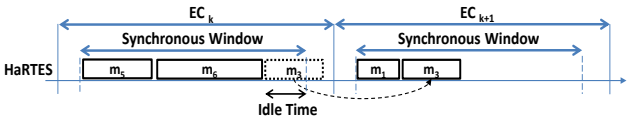

Fig. 3. Inserted Idle Time

IV. Multi-Hop HaRTES Architecture

In this section, we present the multi-hop HaRTES topology and we describe a method, called Distributed Global Scheduling (DGS), to forward the traffic through multiple HaRTES switches. This method was proposed in [4]. In this paper, we compare the new proposed method with the DGS method.

A. Multi-Hop HaRTES Topology

The multi-hop HaRTES architecture is built by connecting multiple HaRTES switches in a tree topology as it presents a good compromise between cabling length and routing complexity. Such a network is depicted in Figure 4.

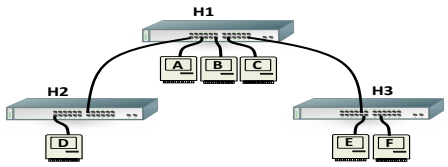

Fig. 4. The Multi-Hop HaRTES Topology

In this architecture we define two types of messages considering their transmission route. Messages that are sent through a single switch are called *local messages*, while messages that are sent through multiple switches are called *global messages*. Moreover, we distinguish the links as follows. The link connected between a node and a switch is called *local-link*, whereas a link between two switches is called *inter-link*.

B. Distributed Global Scheduling

In this method, the scheduling of the messages is carried out with all involved switches. First, the switch, to which the

source node is connected to, schedules the message to be transmitted from the source node and buffers it in its own memory. Then, the second switch in the route of the message schedules the message to be sent from the first switch in a posterior EC. Again, the message is stored in the second switch to be scheduled for the next hop in the next EC. The hop-by-hop scheduling of the message continues until the last switch, where the destination node is connected to. The message is not buffered in the last switch, being immediately forwarded to the destination node in one EC.

All switches have a repository containing the message attributes. In this method, a phase for a message is one of those attributes, which is defined to have different values in each switch. The phase is specified in number of ECs and it determines the time difference between activation of the message in the switch and the activation time in the source node. This parameter is essential to guarantee that, in each switch, the message being forwarded is always received from the previous switch.

Figure 5 illustrates the transmission of a message m_1 from node D, connected to switch H2, to node E, connected to switch H3, in the network depicted in Figure 4. Assume that, the phase for m_1 in switch H2 is 0, while it is 1 and 2 in switch H1 and H3, respectively. When m_1 is activated in EC_k, switch H2 schedules that to be sent and stored in switch H2 itself. The message is activated in switch H1 in the next EC (EC_{k+1}) as the phase is 1, hence switch H1 schedules that to be transmitted from the internal memory of switch H2 to the internal memory of switch H1. In EC_{k+2} the message is activated in switch H3, again since the phase is 2 for the message in switch H3. Also, the destination node, i.e., node E, is connected to switch H3. Therefore, switch H3 schedules m_1 to be sent from the internal memory of switch H1 and forwarded to node E. Note that, the switching delay has no impact in the first two switches as the message is buffered. However, in the last switch the message transmission is affected by the switching delay.

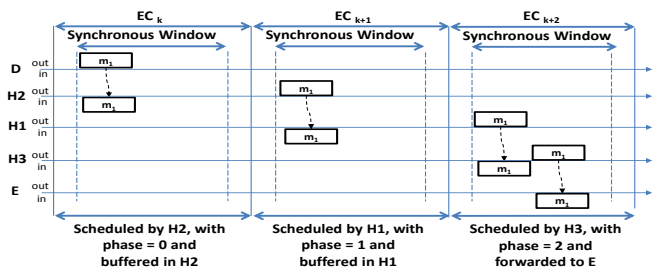

Fig. 5. The Operation of the DGS Method

This method, due to the definition of the phase, requires that the switches are timely synchronized. The synchronization is achieved by the TM transmission from a switch to all nodes and the other switches down the tree topology. In this architecture, the switches synchronize with their parent switch.

Note that, the DGS method was proposed to handle the synchronous messages, only, whereas, the RBS method provides a scheduling method for both synchronous and asynchronous messages.

V. Reduced Buffering Scheme

In this section, we propose a new method, called Reduced Buffering Scheme (RBS), to forward the traffic through multiple HaRTES switches. In order to enable this method on the multi-hop HaRTES architecture, we modified the switch structure, which we first describe. Note that we consider the same multi-hop HaRTES topology depicted in Section IV.

A. HaRTES Switch Structure Modification

In this proposal, we modified the following modules in the HaRTES switch: (i) the scheduler in the Master module, and (ii) the output queues attached to the output ports.

As it can be seen in Figure 1, each output port has three FIFO queues, for synchronous, asynchronous and non-real-time traffic. We modified the queues for synchronous and asynchronous traffic to be priority queues. The non-real-time packets are still forwarded using the FIFO queue.

B. RBS Scheduling Method

In this method all links are partitioned between two windows, synchronous and asynchronous window, the same way as in the DGS method. The allocated window size in each link is differentiated and it is selected based on the actual local and global load crossing that link. Different sizes for the transmission windows increases the efficiency of using the bandwidth in the links. Figure 6 shows the EC partitioning in the RBS method. A particular window (Guard Win) is reserved in the beginning of each EC for different purposes in the RBS method, which is described in this section.

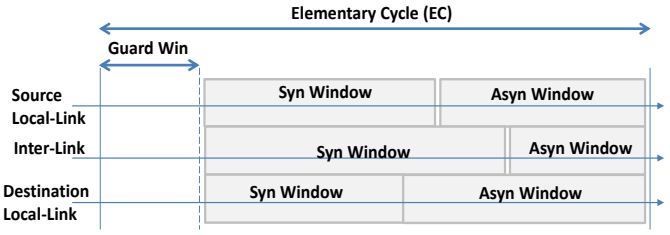

Fig. 6. The EC Partitioning in the RBS Method

The switch to which the source node of a synchronous message is connected to, schedules the message similarly to the single switch case and informs the source node using the TM. The TM is transmitted at the beginning of the EC within the Guard Window (Figure 6).

The source node receives the TM, decodes that and initiates the transmission of the identified messages in the TM. The switch receives the message and inserts it into the priority queue of the output port. If there is enough time within the synchronous window in the current EC, the switch forwards the message to the next switch in the route of the message. However, if the time to transmit the message within the associated window is not enough, the message will be buffered for the next EC. In other words, a message is transmitted through multiple switches as long as there is enough time in the associated window to forward.

Figure 7 shows the same scenario for m_1 presented in Figure 5, yet using the RBS method. In EC_k the message is scheduled by switch H2 to be sent from its source node (node D). Then, it is received by switch H2 and inserted to the priority queue in the output link. As there is still enough time in the synchronous window in EC_k, switch H2 forwards m_1 to switch H1. Similarly to switch H2, switch H1 forwards m_1 to switch H3 as there is still enough time in the synchronous window. However, the transmission of m_1 is suspended in switch H3 due to the lack of remaining time in the synchronous window. In EC_{k+1} the transmission is resumed and m_1 is received by the destination node (node E).

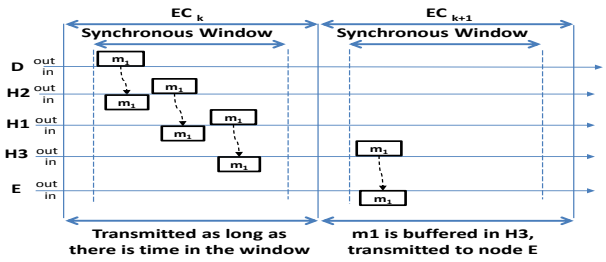

Fig. 7. The Operation of the Reduced Buffering Scheme

The asynchronous messages are forwarded similarly, except that their transmission from their source nodes are autonomous without being triggered by the associated switch. The non-real-time messages are sent within the asynchronous window after transmission of asynchronous messages.

The time synchronization among the HaRTES switches is required to increase the efficiency of the RBS method. For the RBS method, we propose to use a clock synchronization according to the IEEE 1588, which was proposed in [23] for a similar case. The required signaling is carried out within the Guard Window. However, a detailed study of the clock synchronization for the multi-hop HaRTES architecture is out of the scope of this paper.

Comparing RBS and DGS, it is clear that RBS sends the messages faster than that in the DGS method. This is due to the fact that, in the DGS method the messages are buffered in each hop, whereas in the RBS method buffering only occurs when there is not enough time in the current window to continue forwarding. This is achieved thanks to the priority queues in the output ports.

Moreover, the DGS method requires to compute a phase for a message in each switch, which is based on the worst-case response time from the source node to the switch. Applying the worst-case response time for the phases of a message may introduce an unnecessary delay for the message.

VI. System Model

In this paper, we use the real-time periodic model to represent both synchronous and asynchronous messages. The message set, composed by N messages, is defined as follows:

$$\Gamma = \{m_i(C_i, PK_i, D_i, T_i, S_i, Ds_i, P_i, \mathcal{L}_i, n_i), i = 1...N\} \quad (1)$$

In this model, C_i is the total transmission time of the message including its packets. PK_i is the maximum packet size among the packets that compose m_i. D_i is the relative deadline and T_i is the period of m_i, where $D_i \leq T_i$. The period and deadline for the messages are expressed as an integer number of ECs. For asynchronous messages T_i is the minimum inter-arrival time. Moreover, S_i is the source node and Ds_i is the destination node of the message. Currently we restrict our analysis to unicast streams, hence only one destination port per message is considered. P_i denotes the priority of the message. Note that, messages may have the same priority. $\mathcal{L}_i$ is the set of links that m_i passes through, including inter-links, source and destination local-links. Each element in $\mathcal{L}_i$ presents a tuple $l =< x, y >$ which shows a link l between node/switch x to node/switch y. The sequence inside the tuple shows the direction of the message transmission in that link. Also, n_i represents the number of links in $\mathcal{L}_i$, i.e., $n_i = |\mathcal{L}_i|$. The set of links in the route of m_i is presented in (2).

$$\mathcal{L}_i = \{l_k | k = 1...n_i\} \quad (2)$$

Moreover, a set of links which m_i crosses from a specific link l_a until another specific link l_b in its route is defined in (3), where $\mathcal{L}_{i,a,b} \subseteq \mathcal{L}_i$ and $1 \leq a \leq b \leq n_i$.

$$\mathcal{L}_{i,a,b} = \{l_h | h = a...b\} \quad (3)$$

We consider a fixed-priority scheduling policy for the scheduler and we assume that the priority of messages is assigned according to the Rate-Monotonic algorithm.

The switching delay, which is the sum of store-and-forward delay and the hardware fabric latency, is specified by SWD_i.

The total response time for a message m_i is specified by RT_i and it is the time interval between the activation time of the message in the source node and the reception time in the destination node. Moreover, we define the response time of a message m_i crosses the links between link l_a and link l_b as the time duration when the messages is inserted to the priority queue in the switch/node with output link l_a, and the time the message is inserted to the priority queue in the switch/node with input link l_b. This response time is denoted by $RT_{i,a,b}$. Note that both the total response time and the response time between two particular links are expressed in number of ECs. In addition, the idle time is denoted by Id.

VII. Response Time Analysis

According to the RBS method described in Section V, the synchronous and asynchronous messages are transmitted within separated windows, hence they cannot interfere with each other. In this section we develop a response time analysis for both synchronous and asynchronous messages.

A. Response Time Analysis for Synchronous Messages

In the RBS method, a message crosses HaRTES switches in its route until there is not enough time in the transmission window. Then, the message is buffered to be sent in the next EC. In the response time analysis, we capture this behavior by calculating the response time link-by-link from the source node and check whether the message is buffered in any switch connected to that link.

Suppose that, we are calculating the response time of m_1 that is transmitted from node D to node F in the network depicted in Figure 4. First, we compute the response time for the link from node D to switch H2, i.e., the source local-link. We continue to compute the response time for two links from node D to switch H1. Then, we compare the the two computed response times (in number of ECs), i.e., the one for source local-link and the one from node D to switch H1. If they are equal, we continue to calculate the response time for three links from node D to switch H3. However, if they are not equal, we save the response time of the source local-link to the total response time and we start computing the response time from the link between switch H2 to switch H1. We continue the same way until the last link, i.e., the destination (node F) local-link. This way we capture the behavior of the RBS method that has a combination of buffering and forwarding instances for a message through switches. Algorithm 1 illustrates such a calculation.

Algorithm 1 Response Time Calculation for m_i

1: $RT_i = 0$
2: $a = b = 1$
3: **while** $b \leq n_i$ **do**
4: $\quad rt_{i,a,b} = responseTimeCalc(i,a,b)$
5: $\quad RT_{i,a,b} = \left\lceil \frac{rt_{i,a,b}}{EC} \right\rceil$
6: $\quad$ **if** $(a\ != b)$ && $(RT_{i,a,b}\ != RT_{i,a,(b-1)})$ **then**
7: $\quad\quad RT_i = RT_i + RT_{i,a,(b-1)}$
8: $\quad\quad a = b$
9: $\quad$ **else**
10: $\quad\quad b = b+1$
11: $\quad$ **end if**
12: **end while**
13: $RT_i = RT_i + RT_{i,a,(b-1)}$

The algorithm starts by initializing the total response time to zero (line 1). Also, it initializes the links included in the response time calculation to 1 (line 2). Then, the main loop of the algorithm starts to calculate the response time until the last link, i.e., while the condition $b \leq n_i$ is true (line 3). In line 4, the response time of m_i from link l_a until link l_b in the route of m_i is calculated. Whenever both links l_a and l_b are the same, e.g., when they are initialized to 1, the `responseTimeCalc(i, a, b)` in the algorithm calculates the response time of m_i when it crosses just one link, e.g., from one switch to another ($l_a = l_b$). The algorithm scales the response time to the number of ECs to be able to compare that with the previous response time (line 5).

When the algorithm computes the response time in one link (i.e., $l_a = l_b$), the previous response time is not available to compare with the latest response time. In this case, the loop continues for the next link in the route of m_i. This checking is carried out in line 6 and continues in line 9. In contrast, if the response time computation is for several links from link l_a to link l_b, and if the latest response time is not equal to the response time calculated until the previous link (line 6), the message m_i is buffered in the previous switch. Therefore, the algorithm stops calculating and adds up the calculated response time until previous link $l_{(b-1)}$ to the total response time RT_i (line 7). This means that, we calculate the response time for m_i until link $l_{(b-1)}$, where the message is buffered. Then, the algorithm commences to compute the response time from link l_b. Thus, line 8 sets the starting link l_a to the link that the calculation stopped, i.e., link l_b. This procedure continues until the last link in the route of m_i and the algorithm adds up the last response time to the stored total response time during the calculation (line 13).

As it is explained, the function `responseTimeCalc(i, a, b)` computes the response time for m_i from link l_a until link l_b in the route of the message. In this paper, we use the classical response time calculation based on accumulating delays within iterations. However, due to having specified windows for the message transmission, the messages are not allowed to be sent at any time other than their associated window. Therefore, an inflation factor should be taken into account when performing the response time analysis. This issue has been considered previously in [24] and the proposed solution was to inflate the transmission time of the message by a percentage of the bandwidth availability.

Note that, according to the RBS method, the windows size in each link can be different. Also, since Algorithm 1 calculates the response time between two specific links, the inflation factor $\alpha_{i,a,b}$ for m_i should be presented between link l_a and link l_b. Therefore, the inflation factor is calculated in (4) by considering a minimum length of windows in the links between l_a and l_b to assume that the worst-case situation is taken into account. Note that, LW_l is the length of a transmission window in link l_l and $Id_{i,l}$ is the idle time in the transmission windows of link l_l (LW_l).

$$\alpha_{i,a,b} = \frac{\min\limits_{l=a..b}(LW_l - Id_{i,l})}{EC} \tag{4}$$

The idle time is the maximum packet size among the highest and the same priority synchronous messages that share links with m_i in link l_l and the message itself. The idle time is calculated in (5).

$$Id_{i,l} = \max_{\substack{\forall r \in [1,N] \\ \wedge\, m_r \in hep(m_i) \\ \wedge\, l \in \mathcal{L}_r}} (PK_r, PK_i) \tag{5}$$

The response time of m_i, shown in line 4 in Algorithm 1, is evaluated iteratively in (6) by considering the transmission time of the message itself and the interference from other messages. The interference of other messages is categorized in three following parts: (i) the interference from higher and the same priority messages that share links with m_i between link l_a and link l_b, which is specified by $I_{i,a,b}$, (ii) the blocking from the lower priority messages sharing the links with m_i between link l_a and l_b, which is denoted by $B_{i,a,b}$, and (iii) the switching delay of the message that is denoted by $SD_{i,a,b}$. Note that, the interfering and blocking messages are of the synchronous type.

$$rt^{(x)}_{i,a,b} = \frac{C_i}{\alpha_{i,a,b}} + I_{i,a,b} + B_{i,a,b} + SD_{i,a,b} \tag{6}$$

The iteration can start from $x^{(0)} = \frac{C_i}{\alpha_{i,a,b}}$, and the response time of m_i between link l_a and link l_b is calculated in (7).

$$rt_{i,a,b} = rt^{(x)}_{i,a,b} \; when \; rt^{(x)}_{i,a,b} = rt^{(x-1)}_{i,a,b} \tag{7}$$

The first interference term in (6) is caused by the higher and the same priority synchronous messages ($hep(m_i)$) that share links with m_i between link l_a and link l_b in the route of the message. This interference is computed in (8). Note that, the interference should be inflated by the same inflation factor.

$$I_{i,a,b} = \sum_{\substack{\forall j \in [1,N], j \neq i \\ \wedge\, m_j \in hep(m_i) \\ \wedge\, \mathcal{L}_j \cap \mathcal{L}_{i,a,b} \neq \emptyset}} \left\lceil \frac{rt_{i,a,b}^{(x-1)}}{T_j} \right\rceil \frac{C_j}{\alpha_{i,a,b}} \tag{8}$$

According to the RBS method, a message received by a HaRTES switch is inserted to the output priority queue. If there is enough time in the EC, the message is transmitted to the next switch. However, it may happen that, concurrently with the insertion, a lower priority message is transmitted through the same priority queue. Therefore, the arrival message is blocked with the lower priority message. Note that, one particular lower priority message can block m_i only once in the route. Therefore, the same blocking messages in the next links are excluded from the calculation. Also note that, the blocking is at most one packet as the other packets are preempted by m_i. The blocking for m_i is calculated in (9), where $lp(m_i)$ is the set of lower priority synchronous messages than that of m_i.

$$B_{i,a,b} = \sum_{t=a+1..b, a \neq b} \max_{\substack{\forall p \in [1,N] \\ \wedge\, m_p \in lp(m_i) \\ \wedge\, l_t \in \mathcal{L}_p \\ \wedge\, \forall y, a+1 \leq y < t, l_y \notin \mathcal{L}_p}} \left(\frac{PK_p}{\alpha_{i,a,b}} \right) \tag{9}$$

In the RBS method, synchronous messages are transmitted by the source node when indicated in the TM. Therefore, in the source local-link the blocking from the lower priority messages cannot occur. Moreover, blocking may only occur when a higher priority message is received and transmitted within one EC (not buffered). Therefore, a message crossing one link does not cross a switch, hence it is never blocked, i.e., $B_{i,a,b} = 0$ when $l_a = l_b$. Thus, in (9) the summation is performed when $a \neq b$, only. Moreover, the blocking appears at the switch output link. This leads to exclude the first link (l_a) to be accounted for blocking as it is always the input link. Therefore, the summation starts from link $l_{(a+1)}$.

The last term in (6) is the switching delay of the message. As it is described in the system model, the switching delay is the delay of buffering an arrival message before transmitting. This delay is different than the blocking and the interference. The switching delay occurs for a message crossing a switch even without being blocked or delayed by another message. However, the switching delay of other messages that share links with the message under analysis m_i do not affect the switching delay of m_i. In order to show the effect of switching delay of a message on other messages, we consider three different cases that cover all possible scenarios for m_1 and m_2 transmitted through one switch as follows:

Case 1: m_1 and m_2 share an input link, only. In this case the messages have different output links, thus they are stored in the switch independently before transmission. Therefore, the switching delay of each message is equal to its transmission time.

Case 2: m_1 and m_2 share an output link, only. In this scenario the messages are arriving from different source nodes, thus the switch handles them separately and inserts them into the same priority queue. Therefore, the switching delay of each message equals to its transmission time.

Case 3: m_1 and m_2 share both input and output links. In this scenario the first message is stored in the switch and transmitted to the output link. During the transmission of the first message, the second message is being stored and wait for the transmission. Figure 8 shows the transmission of these messages. The switching delay of m_2 is not equal to its transmission time, and in fact it is equal to the transmission time of m_1, assuming $C_1 > C_2$. This is due to the storing time of m_2 which is carried out concurrently with transmission of m_1. Thus, when calculating the switching delay for m_2, the switching delay of m_1 should be considered. This scenario occurs when both messages are transmitted in series from the input to the output of a switch.

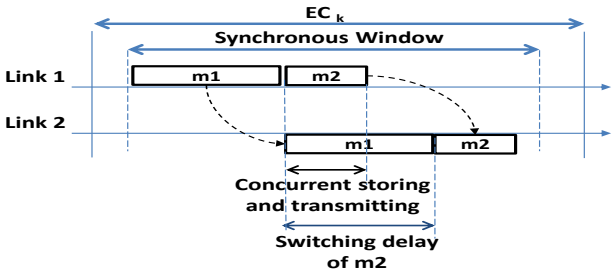

Fig. 8. Switching Delay Example

In general, we can conclude that, the switching delay for m_i is the maximum transmission time between the synchronous messages that share both input and output links with m_i, and the message itself. Note that, all messages including high and low priority messages are taken into account. The reason is that, there might be some lower priority messages ahead of m_i due to the blocking situation in the previous switches. The switching delay for m_i transmitted between link l_a and link l_b is calculated in (10).

$$SD_{i,a,b} = \sum_{t=a+1..b, a \neq b} \max_{\substack{\forall q \in [1,N] \\ \wedge\, l_t \in \mathcal{L}_q \\ \wedge\, l_{(t-1)} \in \mathcal{L}_q}} \left(\frac{SWD_i, SWD_q}{\alpha_{i,a,b}} \right) \tag{10}$$

Note that, when calculating the response time in one link, the message does not cross any switch. Thus, the switching delay does not exist, i.e., $SD_{i,a,b} = 0$ when $l_a = l_b$. Moreover, the same as blocking, the switching delay does not appear in the first link as it is the input link. Thus, the summation starts from the second link $l_{(a+1)}$.

B. Response Time Analysis for Asynchronous Messages

The asynchronous messages are forwarded through multiple HaRTES switches in the same way as the synchronous messages except that they are not triggered by the TM in the source node. Therefore, blocking may occur in the source local-link, in contrast with the synchronous messages, i.e., $B_{i,1,1} \neq 0$. However, in other links than the source local-link, when computing the response time for one link, the blocking

does not exist as the message was buffered and the concurrent transmission with a possible lower priority message did not occur. Thus, $B_{i,a,b} = 0$ when $l_a = l_b \neq l_1$.

In addition, the asynchronous messages are pended during the synchronous window. Thus, they cannot interfere with the synchronous messages. Therefore, when calculating the response time the interfering messages and blocking messages are of the asynchronous type. Also, the inflation factor is calculated considering the asynchronous window for LW.

Algorithm 1 calculates the total response time (RT_i) for an asynchronous message m_i, where the response time between two particular links l_a and l_b are calculated using (6) and (7).

C. Algorithm Complexity

The complexity of Algorithm 1 is $O(N \times M)$ for all messages in a set, where N is the number of messages in the set and M is the maximum number of links in the route of the messages, i.e., $M = \max_{\forall i \in [1,N]}(n_i)$. Moreover, the response time calculation for a message, shown in line 4 of Algorithm 1, is pseudopolynomial. Therefore, the complexity of the algorithm remains pseudopolynomial.

VIII. Evaluation

In this section, we compare the RBS method, proposed in this paper, with the DGS method presented in [4] based on the response time analysis. We show that the RBS method provides a significantly lower response time for the messages compared with the DGS method. In order to present this difference, we defined two networks, a small network consisting of 3 switches and a larger network having 7 switches.

A. Evaluation of a Three-Switches Network

In this section, we considered a network comprising 3 switches along with 6 nodes, as illustrated in Figure 9.

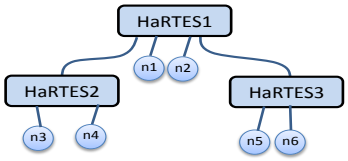

Fig. 9. Three-Switches Network

We generated 50000 sets each containing 20 messages. These messages are generated randomly and all of them are defined as global and of synchronous type to evaluate the method in a multi-hop architecture. The periods of the messages are selected within $[2,22]EC$ and their priorities are assigned based on the Rate Monotonic algorithm within $[1,10]$, where 1 represents the highest priority. Note that, the messages can share a priority level in this example when their periods are equal. Moreover, the transmission time of the messages are chosen within $[80,123]\mu s$, where $123\mu s$ equals to 1542KB and it is the maximum Ethernet packet size. Thus, messages are composed by only one packet in this evaluation.

In addition, the network capacity is set to $100Mbps$ and the hardware fabric latency is $3\mu s$. The EC size was set to $1ms$, where $700\mu s$ are allocated for the synchronous window. Note that, despite the capability of the RBS method to handle different sizes for each link, for the sake of simplicity, we considered the synchronous window in all links to be equal.

In this example, we tagged three messages in each set, a lowest priority, a medium priority and a highest priority. Then, we calculated the response time of the tagged messages according to the DGS and the RBS methods. Note that, we consider only schedulable sets. We compute the difference between the two response times for the tagged messages and we normalize the results using (11).

$$Diff = \frac{RT^{DGS} - RT^{RBS}}{max(RT^{DGS}, RT^{RBS})} \times 100 \tag{11}$$

Furthermore, we counted the sets where the normalized difference of their response times for the tagged messages is within a certain value. These values go from -20% until 90% with an interval of 5%. Note that, the normalized difference cannot reach 100% as the response time in the RBS method cannot be zero in (11). The negative value for $Diff$ shows that the response time for the tagged messages is smaller in the DGS method, whereas the positive value for $Diff$ indicates that the response time for the tagged messages is smaller in the RBS method. Also, the bigger value shows the bigger difference between the response times. Figure 10 depicts an evaluation for the network in Figure 9.

In Figure 10, the x-axis presents the percentage of the normalized difference between the response times in the RBS and the DGS methods for the tagged messages. The y-axis shows the percentage of the sets that have the value within each interval in the x-axis.

As it can be seen, the difference for the highest priority message is never negative. This means, the RBS method conducts the highest priority message always faster than in the DGS method. Moreover, around 46% of the sets have a response time difference within $[50\%,55\%)$ for the highest priority message. In other words, in around half of the generated sets, the highest priority message has around two times better response time in the RBS method compared with the DGS method. Also, in around 46% of the sets, the difference of the response times for the highest priority messages is within $[65\%,70\%)$. The rest of the sets have the other differences, for instance, around 4% have the difference within $[75\%,80\%)$.

The reason for the big difference of the response times for the highest priority message is that, in the DGS method the message is buffered in each switch. However, in the RBS method, the message can be forwarded as long as there is time available in the transmission window. Therefore, the higher priority messages can be conducted through multiple switches as the interference to delay them is very low.

The difference of the response times for the medium priority message is always zero or positive. Around 25% of the sets, the medium priority has within $[0\%,5\%)$ difference of the response times. However, still 12% of the sets have a difference within $[40\%,45\%)$, and 14% of the sets have a difference within $[50\%,55\%)$.

The same for the lowest priority message, the response time in the RBS method is smaller than in the DGS method. This

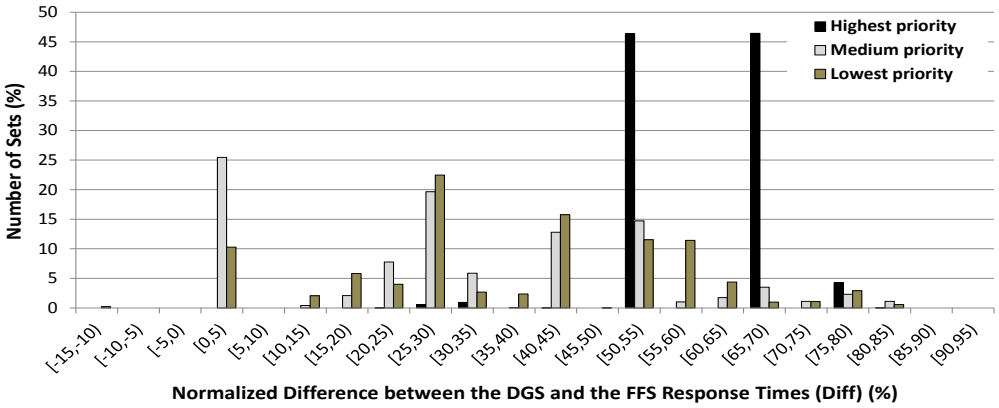

Fig. 10. The Difference between the Response Times in 3-Switches Network

says, even though the lower priority messages are delayed by the higher priority messages, still there might be a chance to cross more than one switch in one EC using the RBS method. Whereas, using the DGS method, the lower priority messages are buffered in all switches in the route of the messages.

Furthermore, we observed $0,21\%$ of the sets having a negative difference $[-15\%,-10\%)$ for the lowest priority message. We will explain the reason for the case where the DGS method gives better results using an example.

Let us assume that m_1 crosses switch H to reach node A in Figure 11, i.e., switch H is the last switch. Also, assume that m_a, m_b and m_c are interfering with m_1 in the input link, and again m_a and m_b are interfering with m_1 in the output link.

In the DGS method, we calculate the response time of m_1 considering m_a, m_b and m_c as interfering messages in both input and output links simultaneously. The reason is, according to the DGS method, the message is not buffered in the last switch and it is forwarded to the destination node immediately when there is enough bandwidth.

Fig. 11. Special-Case Example

In contrast, in the RBS method, according to Algorithm 1, we compute the response time of m_1 link-by-link. Thus, we calculate the response time in the input link considering m_a, m_b and m_c as interfering messages. Then, we compute the response time in the output link taking m_a and m_b as the interfering messages again, that might increase the response time of m_1. This is due to the fact that, in the RBS method, the message can be buffered in the last switch. However, when calculating the response time for the output link, m_a and m_b may not interfere as their effect was already accounted for the input link and they arrived to the destination. This leads to a pessimism in the analysis which is rather complicated to resolve. In fact, the analysis requires to keep track of the interfering messages whether they are interfering. Removing this pessimism remains for future work.

B. Evaluation of a Seven-Switches Network

In this section, we extended the size of the network to seven switches with seven nodes, as depicted in Figure 12.

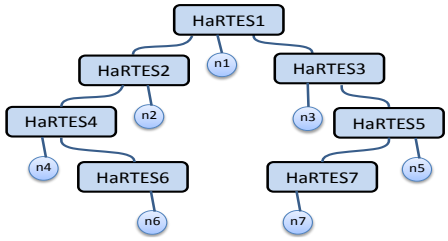

Fig. 12. Seven Switches Network

We generated 50000 sets, each of which containing 30 messages. Similar to the previous section, the messages are generated randomly as global and synchronous type. The properties of the messages are similar to the previous evaluation. Moreover, the network properties are also the same as in the previous evaluation. However, in this example, we set the EC size to $2ms$ and we allocated $1500\mu s$ for the synchronous window equally in all links.

We tagged three messages in these sets, a lowest priority, a medium priority and a highest priority. Then, we computed their response times according to the DGS and the RBS methods. Again, we counted the sets that have a normalized difference of the response times within a certain value. Figure 13 illustrates this evaluation.

As it can be seen, compared to the smaller network with three switches, the difference is bigger, i.e, the bars in the figure are shifted to the right. For instance, around 33% of the sets have now a difference of the response times within $[80\%,85\%)$ for the highest priority message. The reason is that, when we increase the number of switches in the route of a message, the number of occurrence of buffering is enhancing the response time in the DGS method. Whereas, under the RBS method the same message has a chance to cross multiple switches in one EC, in particular for the higher priority messages with lower interference. That is, in fact, the reason for why in Figure 13 the difference of the response times for the highest priority message is always greater than 50%.

Note that, we again observed 0.016% of the sets, where the lowest priority message has a negative difference within $[-15\%,-10\%)$ which is not visible in the figure.

IX. Conclusion and Future Work

In this paper, we have proposed a method, named Reduced Buffering Scheme (RBS), that effectively schedules real-time traffic in multi-hop HaRTES architectures. In order

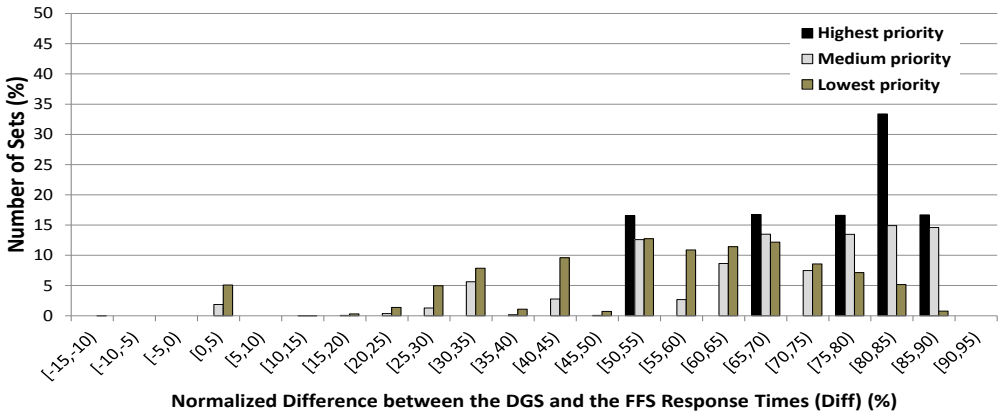

Fig. 13. The Difference between the Response Times in 7-Switches Network

to apply this method for the HaRTES architecture, we have modified the HaRTES switches by changing the output port queue management. The RBS method is based on forwarding messages through switches as long as there is time available in the associated transmission window. Moreover, we have developed a response time analysis for both synchronous and asynchronous messages in the multi-hop HaRTES architecture using the RBS method. Finally, we compared the performance of the RBS method with the Distributed Global Scheduling (DGS) method presented in our previous work by applying the response time analysis of each method on two different networks. We have showed that the response time of the highest priority message is always smaller using the RBS method and the response time of the lowest and medium priority messages can be equal and however are in most of the cases smaller than the case when applying the DGS method. Also, we have showed that this difference increases for larger networks. Our future work aims at implementing the proposed method and experimentally validate the methodology as well as removing some pessimism of the analysis.

ACKNOWLEDGMENT

This work is supported by the Swedish Foundation for Strategic Research via the PRESS project, and the Swedish Knowledge Foundation. Also, it is partially supported by the Portuguese Government through FCT grants Serv-CPS PTDC/EEA-AUT/122362/2010.

REFERENCES

[1] W. Steiner, G. Bauer, B. Hall, M. Paulitsch, and S. Varadarajan, "TTEthernet dataflow concept," in *8th IEEE International Symposium on Network Computing and Applications*, 2009.

[2] Z. Hanzalek, P. Burget, and P. Sucha, "Profinet IO IRT message scheduling," in *21st Euromicro Conf. on Real-Time Sys. (ECRTS)*, 2009.

[3] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida, "Multi-level hierarchical scheduling in ethernet switches," in *Proceedings of the Int. Conference on Embedded Software (EMSOFT)*, October 2011.

[4] M. Ashjaei, P. Pedreiras, M. Behnam, R. J. Bril, L. Almeida, and T. Nolte, "Response time analysis of multi-hop HaRTES ethernet switch networks," in *9th Int. Workshop on Factory Communication Systems (WFCS)*, May 2014.

[5] S. Varadarajan and T. Chiueh, "EtheReal: a host-transparent real-time fast ethernet switch," in *6th Int. Conference on Network Protocols*, 1998.

[6] H. Hoang and M. Jonsson, "Switched real-time ethernet in industrial applications - deadline partitioning," in *9th Asia-Pacific Conference on Communications (APCC)*, 2003.

[7] "IEC 61158, industrial communication networks - Fieldbus specifications," 2010.

[8] I. Land and J. Elliott, *Architecting ARNIC 664 (AFDX) Solutions*, 2011.

[9] "Audio/video bridging task group of ieee 802.1, available at http://www.ieee802.org/1/pages/avbridges.html."

[10] "Powerlink, available at http://www.ethernet-powerlink.org/."

[11] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over COTS Ethernet switches," in *6th IEEE International Workshop on Factory Communication Systems (WFCS)*, June 2006.

[12] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte, "Performance analysis of master-slave multi-hop switched ethernet networks," in *8th IEEE Int. Symp. on Industrial Embedded Systems (SIES)*, June 2013.

[13] A. Mifdaoui, F. Frances, and C. Fraboul, "Performance analysis of a master/slave switched ethernet for military embedded applications," *IEEE Transactions on Industrial Informatics*, 2010.

[14] M. Zhang, J. Shi, T. Zhang, and Y. Hu, "Hard real-time communication over multi-hop switched ethernet," in *The IEEE Int. Conference on Networking, Architecture, and Storage (NAS)*, 2008.

[15] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul, "Methods for bounding end-to-end delays on an AFDX network," in *18th Euromicro Conference on Real-Time Systems*, 2006.

[16] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach," *IEEE Trans. on Industrial Informatics*, 2010.

[17] G. Kemayo, F. Ridouard, H. Bauer, and P. Richard, "Optimistic problems in the trajectory approach in fifo context," in *18th IEEE Conf. on Emerging Technologies Factory Automation (ETFA)*, September 2013.

[18] R. Queck, "Analysis of Ethernet AVB for automotive networks using network calculus," in *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, July 2012.

[19] M. Manderscheid and F. Langer, "Network calculus for the validation of automotive ethernet in-vehicle network configurations," in *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, October 2011.

[20] L. Lenzini, L. Martorini, E. Mingozzi, and G. Stea, "Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks," *Elsevier Performance Evaluation*, vol. 63, October 2006.

[21] J. Schmitt, F. Zdarsky, and M. Fidler, "Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch..." in *The 27th IEEE Conference on Computer Communications*, April 2008.

[22] L. Lenzini, L. Martorini, E. Mingozzi, and G. Stea, "A novel approach to scalable CAC for real-time traffic in sink-tree networks with aggregate scheduling," in *the 1st ACM international conference on Performance evaluation methodolgies and tools*, October 2006.

[23] M. Ashjaei, M. Behnam, G. Rodriguez-Navas, and T. Nolte, "Implementing a clock synchronization protocol on a multi-master switched ethernet network," in *18th Conference on Emerging Technologies Factory Automation (ETFA)*, September 2013.

[24] R. Marau, L. Almeida, P. Pedreiras, K. Lakshmanan, and R. Rajkumar, "Utilization-based schedulability analysis for switched ethernet aiming dynamic QoS management," in *15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010.