J Sign Process Syst
DOI 10.1007/s11265-015-1010-8

Improved Message Forwarding for Multi-Hop HaRTES Real-Time Ethernet Networks

Mohammad Ashjaei[1] · Luis Silva[2] · Moris Behnam[1] · Paulo Pedreiras[2] · Reinder J. Bril[4] · Luis Almeida[1,3] · Thomas Nolte[1]

Received: 15 October 2014 / Revised: 20 February 2015 / Accepted: 21 April 2015

Abstract Nowadays, switched Ethernet networks are used in complex systems that encompass tens to hundreds of nodes and thousands of signals. Such scenarios require multi-switch architectures where communications frequently occur in multiple hops. In this paper we investigate techniques to allow efficient multi-hop communication using HaRTES switches. These are modified Ethernet switches that provide real-time traffic scheduling, dynamic bandwidth management and temporal isolation between real-time and non-real-time traffic. This paper addresses the problem of forwarding traffic in HaRTES networks. Two methods have been recently proposed, namely Distributed Global Scheduling (DGS) that buffers traffic between switches, and Reduced Buffering Scheme (RBS), that uses immediate forwarding. In this paper, we discuss the design and implementation of RBS within HaRTES and we carry out an experimental validation with a prototype implementation. Then, we carry out a comparison between RBS and DGS using worst-case response time analysis and simulation. The comparison clearly establishes the superiority of RBS concerning end-to-end response times. In fact, with sample message sets, we achieved reductions in end-to-end delay that were as high as 80 %.

Keywords Switched Ethernet · HaRTES architecture · Real-time communication · Real-time Ethernet · Response time analysis · Simulation

1 Introduction

Real-time Networked Embedded Systems (NES) are currently found in a vast range of domains, such as industrial process control and supervision, intelligent buildings, energy distribution facilities and cars. In these domains, applications range from embedded command and control systems to image processing, monitoring, human-machine interfacing, etc, that request growing amounts of heterogeneous data to be exchanged over the network, frequently with timeliness constraints and more and more with reconfigurability requirements. This made Ethernet technology particularly appealing for NES, due to features like high bandwidth, openness, wide availability and low cost. Furthermore, it is a mature and well established technology, with millions of nodes installed worldwide, being easy to deploy, manage and maintain. However, issues like a large overhead, lack of connectors suitable to harsh environments and, most notably, an inherently non-deterministic carrier-sense with collision detection (CSMA/CD) contention resolution mechanism, conditioned the adoption of Ethernet in time-critical applications for many years.

To overcome those shortcomings several modifications, extensions or limitations, have been introduced to enforce time-constrained communication over Ethernet, originating the so called Real-Time Ethertnet (RTE) protocols [1], e.g.,

✉ Mohammad Ashjaei
mohammad.ashjaei@mdh.se

[1] MRTC/Mälardalen University, Västerås, Sweden

[2] DETI/IT/University of Aveiro, Aveiro, Portugal

[3] IT/DEEC/University of Porto, Porto, Portugal

[4] Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands

TTEthernet [2], PROFINET [3] and more recently Ethernet AVB [4–6].

Several strategies have been followed by existing RTE protocols, typically favoring a certain aspect such as safety, timeliness, heterogeneity or reconfigurability in dynamic real-time applications. Dynamic reconfigurability is specially important to improve efficiency in network bandwidth usage, particularly in bandwidth consuming industrial media control applications [7], such as machine vision [8], automated inspection [9], vehicle guidance [10], and even infotainment applications in vehicles [11, 12]. In these applications, bandwidth usage can be traded online for quality of service (QoS), e.g., to serve more multimedia streams with reduced QoS or improve the QoS when there are less streams and there is bandwidth available. However, ensuring a continued adequate QoS requires an appropriate management that enforces continued timeliness.

Protocols catering for all these requirements are still very few. This is the case of the Hard Real-Time Switching architecture (HaRTES) [13] that aims at supporting dynamic and heterogeneous communication requirements with efficient bandwidth management in NES. HaRTES supports real-time periodic, real-time sporadic and non-real-time traffic with temporal isolation between each type of traffic, hence addressing the heterogeneity requirement. In the HaRTES architecture, the former traffic type is called *synchronous traffic*, while the two latter types are called *asynchronous traffic*. Further, the traffic is scheduled dynamically, thus its properties can be updated during system operation in order to adapt to eventual changes on the application requirements. Finally, it has integrated bandwidth management with an admission control mechanism that assures a continued real-time behavior of the system, even during reconfigurations.

1.1 Problem and Contributions

This work addresses the problem of building large networks with multiple interconnected HaRTES switches extending their individual properties to end-to-end connections. This problem was originally addressed in [14] where a traffic forwarding method was proposed, called Distributed Global Scheduling (DGS). However, DGS produces high end-to-end response times due to buffering them between switches.

Recently, we proposed a new forwarding method called Reduced Buffering Scheme (RBS) [15] that significantly reduces the end-to-end response times of DGS and we also provided a worst-case response time analysis. Moreover, we compared RBS and DGS based on the worst-case response time analysis. In this paper we extend such work with the following contributions:

1. Design and implementation of RBS within the HaRTES architecture;
2. Comparison between RBS and DGS with respect to end-to-end response times based on simulation;
3. Experimental validation of RBS with an implementation prototype.

1.2 Overview

The rest of the paper is organized as follows. The next section discusses the related work. Section 3 presents the HaRTES architecture, while Section 4 presents multi-hop HaRTES architecture including DGS and RBS forwarding methods. Section 5 presents the system model that is used in Section 6 for the response time analysis. Then, Section 7 shows the comparison between RBS and DGS. Section 8 presents the hardware modifications, while Section 9 shows the experimental validation on hardware. Finally, Section 10 concludes the paper and presents some directions for future work.

2 Related Work

In this section we present an overview of RTE protocols and their timing analysis.

2.1 Overview of RTE Protocols

Research on the use of Ethernet for real-time applications dates back to more than 30 year ago. For shared Ethernet, see the survey of earlier RTE protocols [16], and for switched Ethernet, EtheReal [17] and the EDF scheduled switch [18], which are both based on channel reservations support on enhanced switches.

More recently, many academic and industrial RTE protocols have been actively developed most of which reached the market. Some of the more prominent ones are TTEthernet [2], PROFINET IRT [3], Ethernet POWERLINK [19], EtherCAT [20] and Atacama [21]. All these protocols are based on the time-triggered (TT) communication paradigm, complemented with some support to event-triggered (ET) traffic. Moreover, TT traffic is statically configured, at pre-runtime and cannot be changed online.[1] At best there is support to a pre-defined set of modes that can be selected online. Therefore, these protocols are not suitable to handle dynamic real-time applications.

Avionics Full DupleX Switched Ethernet (AFDX) is a standard that defines the electrical and protocol specifications for the data exchange between Avionics Subsystems.

[1]EtherCAT does not exclude run-time traffic updates but it is essentially used with static communication requirements

It is based on standard Ethernet switching components that performs the standard functions, such as filtering, policing, and forwarding. However, some services are statically configured to assure a deterministic behavior. AFDX networks are based on the concept of Virtual Links (VL), which define the logic path between source and destination end-system(s). The load submitted by each VL is limited thanks to the imposition of the so-called Bandwidth Allocation Gap between successive frames. When properly designed, end-to-end delays can be upper bounded.

More recently, Audio Video Bridging (AVB), a set of technical IEEE standards ([4–6]), is gaining momentum, mainly among the automotive industry. For example, BMW is using this technology since 2013, in its X5 model. Ethernet AVB supports IEEE 1588-based clock synchronization, bandwidth reservations and traffic shaping services, at switches and end nodes. AVB defines currently two traffic classes (named **A** and **B**), and for seven hops, guarantees a maximum delay of 2ms and 50ms, respectively.

Both AVB and AFDX are inherently event-triggered and offer predictable services, with upper bounded latency. In principle, it would be possible to build dynamic systems, although at the moment of this writing we could not find references to that in the literature. However, they lack support to scheduled traffic, and thus low-latency and low-jitter traffic, common in many real-time applications, is not supported. Recently, an effort from the academia to add scheduled services to AVB has been done [22]. According to this proposal, a higher priority class above class A is introduced such that the traffic of that class does not go through the traffic shaper. This type of traffic is called Scheduled Traffic (ST) and its transmission is planned offline. However, at this moment it is not accepted as part of the Ethernet AVB standard.

The IEEE 802.1 Time Sensitive Networking Task Group [23] is continuing the work previously developed in the scope of Audio Video Bridging (AVB), developing a set of technical IEEE standards for Ethernet networks that define mechanisms for transmission of low latency data and to provide high availability. Potential applications include industrial control and supervision, as well as vehicular networks. In particular, 802.1Qbv controls data transmission on a class-level basis, instead of a per-stream level. As classes are defined by the priority of the VLAN tag, the maximum number of priorities is 8, which limits the system schedulability. Despite presenting interesting improvements, at the scheduling level, IEEE 802.1Qbu/Qbv have limited schedulability due to the reduced number of priorities. Also, the time-aware shapers are strictly periodic, that causes problems with data sources having non-harmonic periods. Moreover, the standards do not define how these shapers are managed, so it is not clear if orchestrated online updates of multiple shapers can be made efficiently.

From the above discussion we can conclude that RTE protocols either favor static TT services, or adopt pure ET approaches. In both cases, they are not suitable to cope with the requirements of emerging dynamic applications. In the former case the protocols are too rigid, not allowing dynamic adjustments of the communication subsystem to the application requirements. In the second case the protocols cannot guarantee short response times to TT traffic. This observation led originally to the development of the FTT-SE protocol [24], which features online scheduling in a master node, natural support to TT and ET traffic types and integrated QoS and admission control. As FTT-SE is based on COTS switches, it presents some structural limitations and vulnerabilities, the most important one is relying on the end nodes for controlling the ingress traffic. Consequently, for instance, the presence of a single node not compliant with the protocol (or actually a malfunction FTT-compliant node) endangers the timeliness guarantees of all traffic.

The HaRTES switch [25] overcomes this problem by inserting the master module inside the switch and by using the information provided by the master to filter incoming packets. Thus, the HaRTES switch enforces traffic confinement and rejects packets that violate the associated reservations.

Despite the similarities between FTT-SE and HaRTES, there are also subtle but important differences, which have a strong impact on 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 used in the FTT-SE, HaRTES has the ability to selectively buffer the traffic. Therefore, the results previously developed for multi-hop communication in the FTT-SE networks (e.g. [26]) would result in sub-optimal performance.

A qualitative comparison of the RTE protocols mentioned above is given in Table 1.

2.2 Timing Analysis Approaches

For real-time network applications, a timeliness guarantee is demanded. For such applications a worst-case response time analysis is required to provide analytical bounds on the maximum delay that communication can take.

Regarding the timing analysis of multi-hop Ethernet networks, several methods are utilized. In the work presented in [27], 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. In addition, Network Calculus is used in networks of standard Ethernet switches as presented in [28]. In [29] three methods are used to derive the end-to-end traffic delays in a multi-hop AFDX network. These three methods include Network

Table 1 Qualitative comparison among RTE protocols.

RTE Protocol	Time-Triggered	Event-Triggered	Clock Synchronization	Online Reconfiguration for Time-Triggered Traffic
TTEthernet	Yes	Yes (rate constrained)	Requires	No
PROFINET IRT	Yes	Yes (limited)	Requires	No
Ethernet POWERLINK	Yes	Yes	Provides	No (but possible)
EtherCAT	Yes	Yes	Provides	No (but possible)
EtheReal [17]	No	Yes	No	Not applicable
Atacama [21]	Yes	Yes	Requires	Yes (predefined)
EDF Scheduled Switch [18]	Yes	Yes	Requires	Yes
AFDX	No	Yes	No	Not applicable
Ethernet AVB	Under modification	Yes	Requires	Yes
FTT-SE [24]	Yes	Yes	May provide	Yes
HaRTES [25]	Yes	Yes	May provide	Yes

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 [30]. However, in [31], 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. The solution to solve the optimism problem is presented recently in [32].

In the context of Ethernet AVB, the work presented in [33] has utilized the Network Calculus method to derive traffic end-to-end delays. Also, the work in [34] 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. Recently, a schedulability analysis based on a response time calculation for the traffic class A and B is proposed in [35].

A different approach is followed in [36] and [37] that derives 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 [38] utilized the mentioned method to investigate an admission control in sink-tree networks.

In our previous work, presented in [26], we computed the worst-case end-to-end delay of traffic, based on a response time analysis, for multi-hop communication in the FTT-SE protocol and we compared the results with the ones computed using Network Calculus. We showed that, Network Calculus generates higher pessimism. Thus, we present a response time analysis in line with [26] but adjusted to the Reduced Buffering forwarding Scheme. Note that no other existing switched Ethernet architecture presents the whole set of properties exhibited by HaRTES-RBS which, for that reason, required a specific analysis.

3 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.

3.1 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 Fig. 1. The packets arriving at the input ports are analyzed by the Packet Classification module, which is implemented for each input port. This module separates 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. Note that, HaRTES handles arbitrary large messages, thus these messages are fragmented to small packets.

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 3.2.

For each output port three FIFO queues are implemented to handle synchronous, asynchronous and non real-time

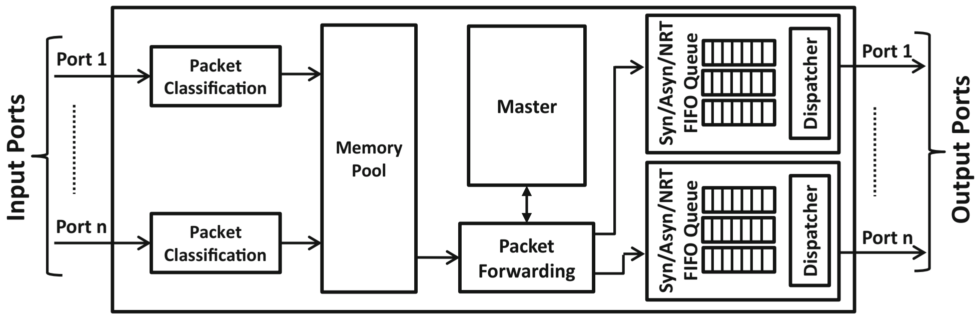


Figure 1 HaRTES functional structure.

packets, which are identified as Syn, Asyn and NRT respectively in Fig. 1. The dispatcher allows packet transmission from each queue during the associated windows only, hence it handles the transmission in the reserved bandwidths, 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(s) 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 traffic.

3.2 HaRTES Traffic Scheduling

The HaRTES architecture is a micro-segmented network that is 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 in two windows, one for scheduling the synchronous traffic (Synchronous Window) and the other one for asynchronous traffic (Asynchronous Window), as shown in Fig. 2. Note that the Asynchronous Window is used for both asynchronous and non-real-time traffic. The latter is transmitted strictly after the former and when there is enough time left free in the Asynchronous Window.

In each EC the switch determines new activations of the synchronous messages, it updates the ready queue and checks which 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 (Fig. 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 [13]. 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 delays, which are known as *store-and-forward* and *hardware fabric latency*. The former corresponds to the time required to receive a 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, the transmission of messages that cannot be fully transmitted within the transmission window is delayed for the next EC, e.g., m_3 in Fig. 3. This behavior introduces an idle time in each transmission window.

4 Multi-Hop HaRTES Architecture

In this section, we present the multi-hop HaRTES topology and we describe two methods, called Distributed Global Scheduling (DGS) and Reduced Buffering Scheme (RBS), to forward the traffic through multiple HaRTES switches. DGS was proposed in [14] and RBS was presented in [15].

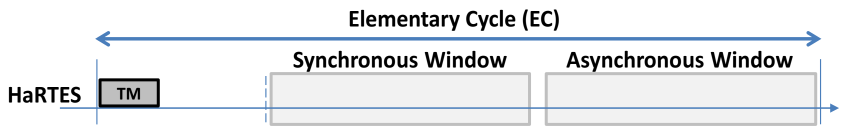


Figure 2 The EC partitioning in HaRTES.

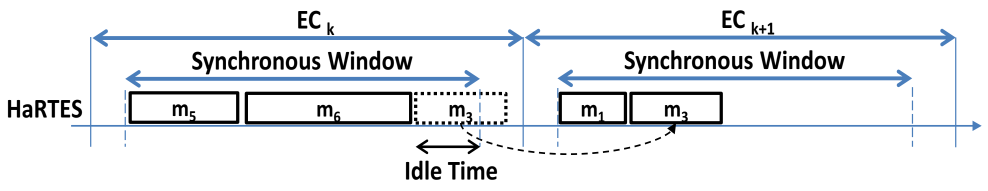


Figure 3 Inserted idle time.

4.1 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. Figure 4 shows an example of a HaRTES network, where H1-H3 are the HaRTES switches, responsible to schedule the traffic, and nodes A-F represent data sources and/or consumers (e.g., sensors, controllers, actuators), that will exchange real-time data via the HaRTES network.

In this architecture we define two types of messages based on their transmission route. Messages that are sent through a single switch are called *local messages*, while messages that cross 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*.

4.2 Distributed Global Scheduling

In this method, the scheduling of the messages is carried out by 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 message m_1 from node D, connected to switch H2, to node E, connected to switch H3, in the network depicted in Fig. 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 it 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 it 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 by 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.

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 to the other switches down the tree topology. In this architecture, the switches synchronize with their parent switch.

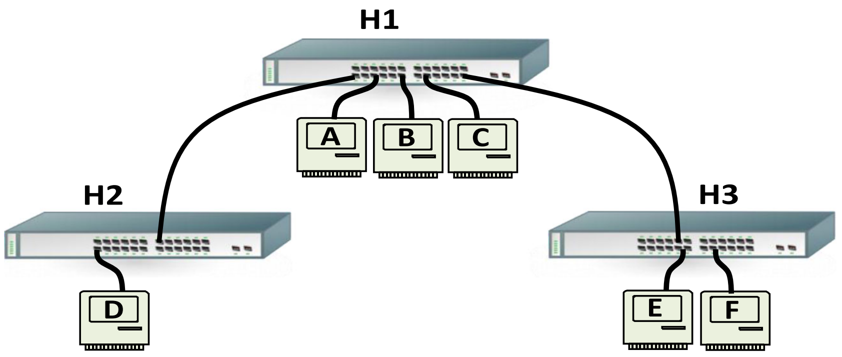


Figure 4 The multi-hop HaRTES topology.

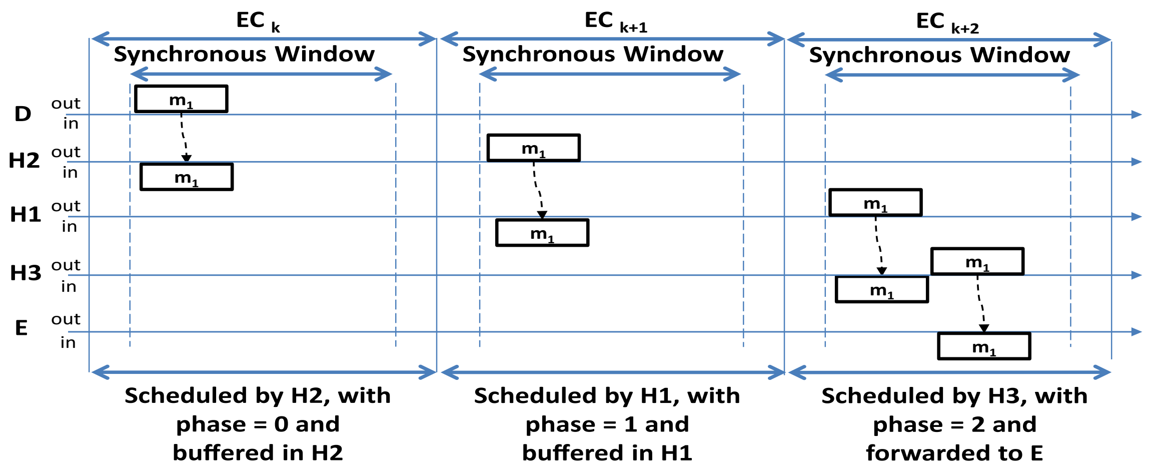

EC k
EC k+1
EC k+2
Synchronous Window
Synchronous Window
Synchronous Window
D
H2
H1
H3
E
out
in
m1
Scheduled by H2, with phase = 0 and buffered in H2
Scheduled by H1, with phase = 1 and buffered in H1
Scheduled by H3, with phase = 2 and forwarded to E

Figure 5 The operation of the DGS method.

Moreover, in order to minimize the end-to-end delay, the DGS method requires that each switch schedules the global messages with a given relative offset, which is based on the worst-case response time from the source node to each one of the switches in the path. This approach, in general, entails some degree of pessimism leading to longer maximum end-to-end delays than necessary. Among the sources for pessimism are the blocking and interference levels that are typically lower than assumed by the analysis.

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.

4.3 Reduced Buffering Scheme

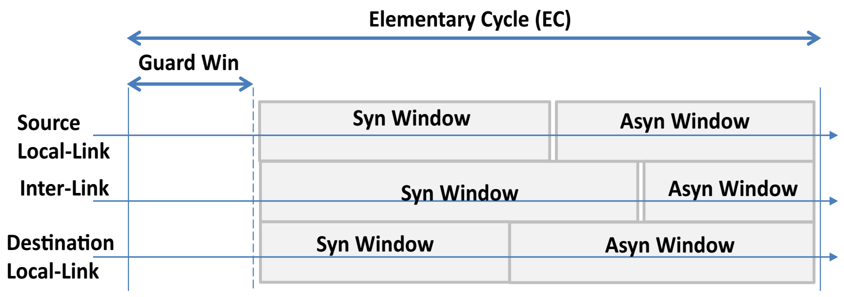
The DGS method is conceptually very simple. In fact the interlinks are handled in the exact same way as the local links. However, such simplicity comes at a price. In this particular case each hop in the network causes a delay of at least one cycle. In reality, such hop delay can often be much higher than one EC, due to the need to accommodate the scheduling jitter. Therefore, the schedulability of multi-hop message streams may degrade severely when the network dept grows and the number of messages increases.

This performance bottleneck can be tackled by changing the way traffic is handled at the output ports. In fact, source local-links have a structure and operation methodology that is severely constrained by the need to control the ingress traffic. Interlinks and destination local-links do not have such constrains, since the traffic that crosses them was already scheduled (if synchronous) and validated (if real-time) by the first HaRTES switch in the chain. Therefore, we advocate the use of simple priority queues at the output ports, combined with a dispatcher to enforce the EC phases. In this way global messages are forwarded to the next hop as soon as they reach the top of the queue and, of course, when the active window is the correct one.

As shown in Fig. 6, the RBS scheduling model maintains the EC organization, namely the partitioning between synchronous and asynchronous windows. However, the window size can be individually adjusted in each interlink, to accommodate the balance between local and global load crossing that link. As for local links, a Guard Window is also reserved in the beginning of each EC, for management purposes.

The uplink between the source node and the first HaRTES switch in the multi-hop chain is managed in the standard way. This means that the switch schedules the synchronous messages and sends the EC schedule (via the TM) to the slave node(s), at the beginning of the EC within the

Figure 6 The EC partitioning in the RBS method.



guard window. Then, slave nodes decode the TM and, when they are producers of scheduled messages, initiate their respective transmission within the synchronous window.

Messages are placed in the priority queue of the output ports. Concurrently, the dispatcher checks for the EC phase and, during the synchronous window, continuously forwards messages waiting on the priority queue, if any. Since these messages come from an interlink the next switch in the chain receives the global messages and immediately places them on its own interlink priority queue. The process is iterated until the end of the synchronous window. Whenever that happens, the messages in the priority queues are put on hold, by the dispatcher, until the synchronous window of the following EC, when the whole process is repeated. It can be readily seen that this strategy, which consist in allowing each switch to forward global messages autonomously, potentially reduces the hop delay. In fact, high priority messages may easily cross several hops in a single EC, a situation that was impossible in DGS. Local messages, after being received by the switch, are inserted to the priority queue of the destination local-link(s). The same way as global messages, local messages are transmitted to the destination when there is enough time in the respective window.

To illustrate the difference between both global traffic forwarding methods, Fig. 7 shows the transmission of message m_1 using the RBS method. The set-up is the same as the one shown in Fig. 5, except that in this latter case it was used the DGS method.

In EC_k the message is scheduled by switch H2. Then, its source node (node D) sends it to switch H2, which inserts it in the priority queue of the interlink. As message m_1 is received before the end of EC_k's synchronous window, 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).

Asynchronous real-time messages are forwarded according to the RBS method, exactly in the same way as the synchronous ones. The only difference is that they are sent by the source nodes autonomously, typically in response to some changes in the environment, instead of being scheduled by the switch (Master). Finally, the non-real-time messages are sent within the asynchronous window, after transmission of all asynchronous messages, in a way that approaches the operation of a background server. It should be stressed that this class of traffic is handled strictly in FIFO order and without timeliness guarantees (best-effort service class).

Time synchronization among the HaRTES switches allows increasing the efficiency of the RBS method. We propose the use of an IEEE 1588 based clock synchronization mechanism, as the one proposed in [39] for a similar case. The required signaling is carried out within the Guard Window. A detailed study of the clock synchronization for the multi-hop HaRTES architecture is out of the scope of this paper.

Finally, several faults can affect the performance of the network, e.g., TM missing, master fault and message missing. Mitigating such faults requires error detection and recovery mechanisms such as those proposed in [40, 41]. Fault tolerance issues are, however, out of the scope of this paper.

5 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 in expression (1). The parameters are explained in Table 2.

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

In the analysis, we consider constrained deadline, i.e., $D_i \leq T_i$. Also, The period and deadline for the messages are expressed as an integer number of ECs. As it is mentioned in Table 2, $\mathcal{L}_i$ represents a set of links that m_i traverses. Each

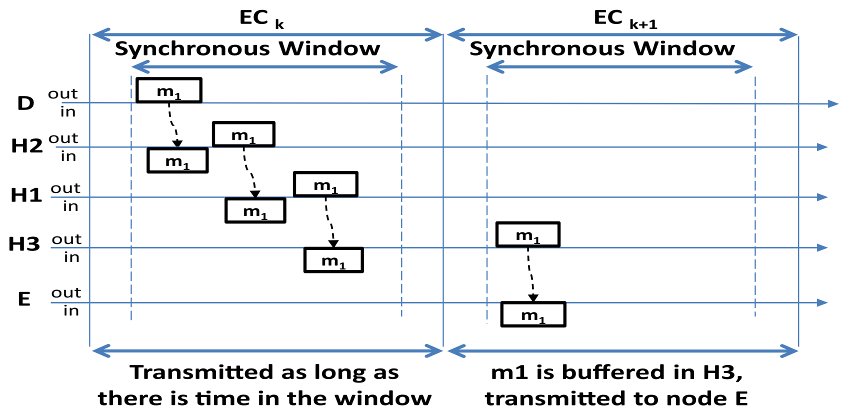

EC k
Synchronous Window
EC k+1
Synchronous Window
D
H2
H1
H3
E
out
in
m1
Transmitted as long as there is time in the window
m1 is buffered in H3, transmitted to node E

Figure 7 The operation of the reduced buffering scheme.

Table 2 The notations used for the message parameters.

Notation	Description
C_i	The total transmission time of m_i including its packets.
PK_i	The maximum packet size among the packets that compose m_i.
D_i	The relative deadline of m_i.
T_i	The period of m_i. It also represents the minimum inter-arrival time for asynchronous messages.
S_i	The source node of m_i.
Ds_i	The destination node of m_i.
P_i	The priority of m_i.
$\mathcal{L}_i$	The set of links that m_i traverses including inter-links, source and destination local-links.
n_i	The number of links m_i crosses through, i.e., $n_i = \lvert\mathcal{L}_i\rvert$.
l_x	It represents the link number x in the network.
EC	The elementary cycle size.
LW_l	The synchronous window size in link l.
RT_i	Response time of m_i.
$RT_{i,a,b}$	Response time of m_i crossing from link l_a to link l_b.

member of $\mathcal{L}_i$ is presented by 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. The set of links in the route of m_i is presented in expression (2). Moreover, we restrict the analysis to unicast streams, hence only one destination port per message is considered. Note that multicast/broadcast streams can be handled by converting them in multiple unicast streams as adequate, but we leave this case out of this paper for the sake of clarity.

$$\mathcal{L}_i = \{l_k | k = 1...n_i\} \tag{2}$$

Moreover, the 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 Eq. 3, where $\mathcal{L}_{i,a,b} \subseteq \mathcal{L}_i$ and $1 \leq a \leq b \leq n_i$.

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

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. Note that messages may have the same priority when their periods are equal.

The switching delay, which is the sum of the hardware fabric latency and the store-and-forward delay, 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 that crosses the links between link l_a and link l_b as the time lapse between the time 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.

6 Response Time Analysis for RBS

In this section, we briefly discuss the analysis for RBS for the sake of completeness. For more details the reader is referred to [15]. Note that the response time analysis for a single-switch HaRTES architecture is a special case of the analysis for the multi-hop HaRTES architecture, i.e., the presented response time analysis applies to both the single-switch and the multi-hop HaRTES architecture.

According to the RBS method, the synchronous and asynchronous messages are transmitted within separated windows, hence they cannot interfere with each other. Therefore, we discuss the response time analysis for them separately.

6.1 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 A to node B in the network illustrated in Fig. 8.

First, we compute the response time for the link from node A to switch H1, i.e., the source local-link (RT_1). Second, we compute the response time for two links from node A to switch H2 (RT_2). Then, we compare the two computed

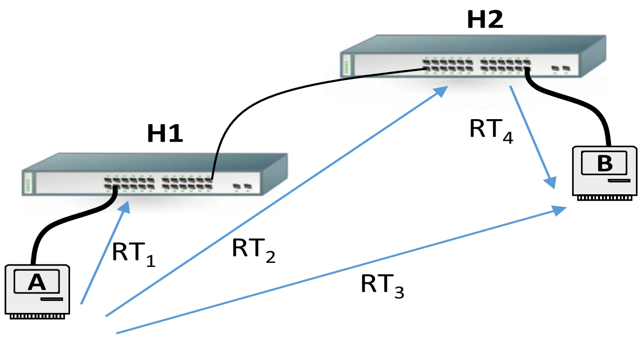


Figure 8 Example of response time analysis procedure.

response times (in number of ECs), i.e., the one for source local-link and the one from node A to switch H2. If they are equal, which means that the message is transmitted in the same EC, we continue to calculate the response time for the three links from node A to node B (RT_3). Again, we compare the new computed response time with the previous one. In this example, assume that RT_2 and RT_3 are not equal. This means that the message is buffered in the switch H2, hence we store the previous response time (RT_2) to the total response time and we start computing the response time from the link between switch H2 and node B (RT_4). Finally, the total response time, in this example, is the sum of the stored total response time and the new computed response time, i.e., $RT_2 + RT_4$. 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:     $rt_{i,a,b} = responseTimeCalc(i, a, b)$
 5:     $RT_{i,a,b} = \left\lceil \frac{rt_{i,a,b}}{EC} \right\rceil$
 6:     if $(a\ != b)\ \&\&\ (RT_{i,a,b}\ != RT_{i,a,(b-1)})$ then
 7:         $RT_i = RT_i + RT_{i,a,(b-1)}$
 8:         $a = b$
 9:     else
10:         $b = b + 1$
11:     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 it 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.

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 Eq. 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 Eq. 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 Eq. 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.

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

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 Eq. 7.

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

The first interference term in Eq. 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 Eq. 8. Note that, the interference should be inflated by the inflation factor of the associated link.

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

According to the RBS method, a message received by a HaRTES switch is inserted into 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, at most one lower priority packet can block m_i and only once in the route. The blocking for m_i is calculated in Eq. 9, where $lp(m_i)$ is the set of lower priority synchronous messages than that of m_i.

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

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 Eq. 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 Eq. 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. 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.

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 ones, are taken into account. The switching delay for m_i transmitted between link l_a and link l_b is calculated in Eq. 10. For more details on the proofs, the reader is referred to [15].

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

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 $l_{(a+1)}$.

6.2 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 links other 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, when calculating the response time the interfering 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 Eqs. 6 and 7.

6.3 Single-Switch Response Time Analysis

The response time analysis for the single-switch HaRTES architecture is essentially a specific case of the analysis for the multi-hop HaRTES architecture. In a single-switch network where the traffic is transmitted using the RBS method, a message crosses at most one HaRTES switch, hence two links, i.e., the source and destination local-links. Therefore, in Algorithm 1, a and b become the source local-link and the destination local-link, respectively. Moreover, the computation, that is presented in Eqs. 6 and 7, maintains the same. However, in the blocking and switching delay calculations, which are presented in Eqs. 9 and 10, the summation will be removed as a message is crossing one switch, only. This means that there is just one blocking message in the destination local-link, and one switching delay caused by at most one message.

6.4 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 pseudo-polynomial as it is a recursive function. Therefore, the complexity of the algorithm remains pseudo-polynomial.

7 RBS and DGS Comparison

In this section, we evaluate the RBS method in different perspectives. First of all we compare the RBS method, proposed in this paper, with the DGS method presented in [14] based on the response time analysis. We show that the RBS method decreases response time of messages significantly compared with the DGS method. We present this comparison in two different network sizes, a small network consisting of 3 switches and a larger network with 7 switches.

In our second evaluation, we define a network with a set of messages that are generated randomly. Then, we simulate the traffic transmission in the RBS and DGS methods. We measure and calculate the messages response times in both methods and we show the differences from calculation and simulation perspectives. In the third evaluation, we generate 50 random network architectures with random set of messages and we simulate the traffic forwarding in both methods. Again, we show the differences of the measured response times, in the generated architectures.

In all evaluations in this section, we set the network capacity to $100 Mbps$ and the hardware fabric latency of the switch is $3\mu s$. Also, we assumed that all messages are composed by only one packet. 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.

7.1 Comparison based on the Response Time Analysis

For this comparison we defined two different network sizes, one with three switches and the other one with seven switches.

7.1.1 Scenario 1: Three-Switches Network

We considered a network comprising 3 switches along with 6 nodes, as illustrated in Fig. 9.

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. The periods of the messages are selected within $[2, 22]EC$, and their priorities are assigned based on the Rate Monotonic algorithm. Note that the messages share a priority level when their periods are equal. Moreover, the transmission time of the messages is chosen within $[80, 123]\mu s$.

In addition, the EC size was set to $1ms$, where $700\mu s$ are allocated for the synchronous window.

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 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 Eq. 11.

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

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 Eq. 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

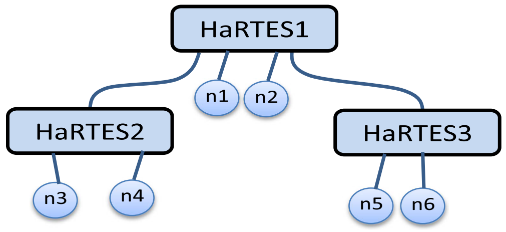


Figure 9 Three-switches network.

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 the result of the evaluation for the network in Fig. 9.

In Fig. 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 that 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. For 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 %).

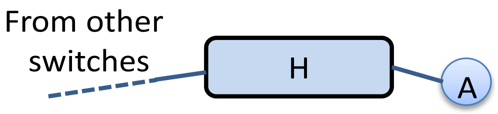


Figure 11 Special-case example.

The same happen for the lowest priority message. The response time in the RBS method is smaller than in the DGS method. This says that 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 have 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 Fig. 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.

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

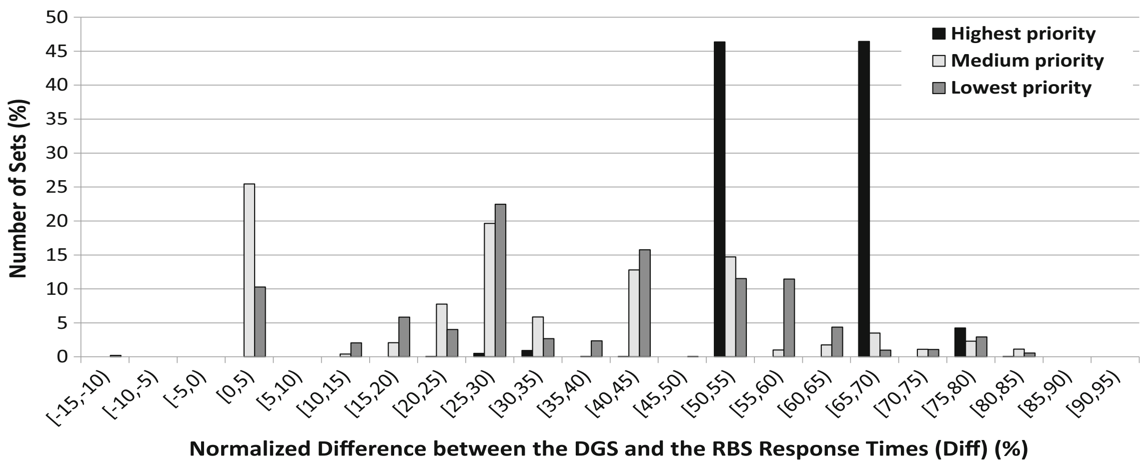


Figure 10 The difference between the response times in 3-switches network.

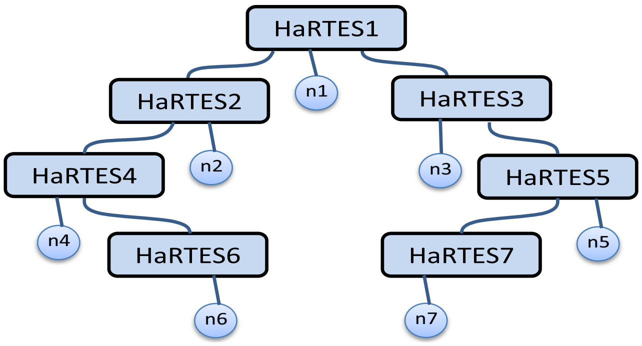


Figure 12 Seven switches network.

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 solve. In fact, the analysis requires to keep track of the interfering messages whether they are interfering. Removing this pessimism remains for future work.

7.1.2 Scenario 2: Seven-Switches Network

We extended the size of the network to seven switches with seven nodes, as depicted in Fig. 12.

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

As the previous experiment, 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 why in Fig. 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.

7.2 Comparison based on Simulation

In this section we assumed a network example with three switches and six nodes as shown in Fig. 9. We generated a set of 20 messages with the following parameters. We selected the priorities of the messages based on RM policy. The messages can share a priority level. Also, the periods of the messages are selected within $[5, 25]ECs$ and the deadlines are chosen to be equal to the periods. As the RBS and the DGS methods are developed to handle mainly the global messages, the generated messages in this example are all synchronous and global. All 20 generated messages in this simulation contain one packet, only, with transmission time of $123\mu s$, i.e., 1500bytes payload. In this example, we set

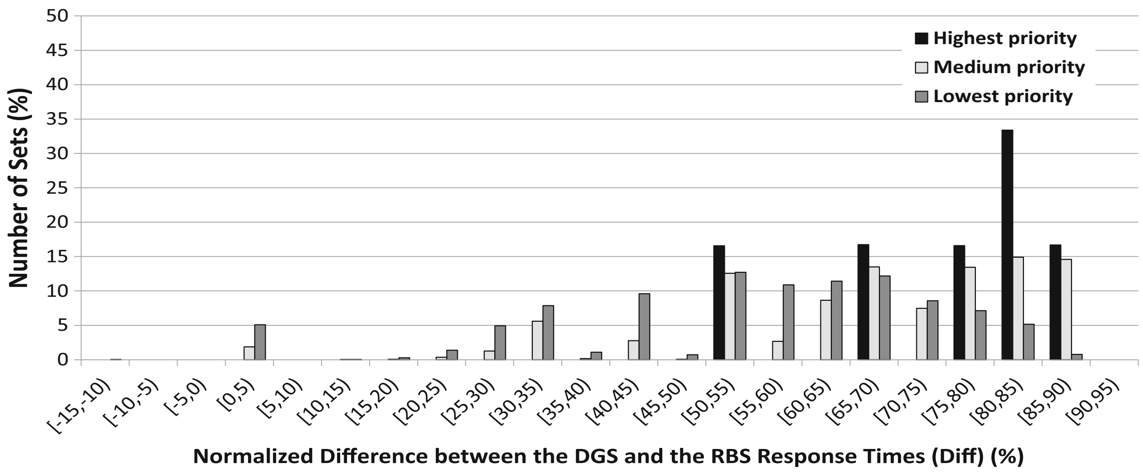


Figure 13 The difference between the response times in 7-switches network.

the EC to $1ms$ where we allocated $0.7ms$ to the synchronous window in both methods.

In order to simulate the example we have used the simulation tool presented in [42]. However, the simulation tool was initially developed to support the FTT-SE architectures. Thus, we modified the tool, first to support the DGS method in the multi-hop HaRTES architecture in [14], then we also modified that to support the RBS method. These modifications were simplified by the fact that the tool is developed in Simulink with a modular kernel.

Table 3 shows the messages parameters including the period (T) and priority (P). We show the maximum response time of the messages measured in the DGS (Sim-DGS) and the RBS methods (Sim-RBS) in the simulation. We also show the calculated response times using the analysis presented in this paper for the RBS method (RT-RBS), and the analysis presented in [14] for the DGS method (RT-DGS). We simulated the example for 50000 ECs. Note that in the table the periods, measured response times and calculated response times are presented in number of ECs.

As we can see from the results, the calculated response times in both methods are always higher or equal to the measured response times. The computed response times in the RBS method are always lower than the DGS method, except m_6 where both calculated response times are equal. m_6 has priority of 4 and according to the previous evaluation there are cases from the medium and low priority messages that have the same response times.

Table 3 The messages respone time for the RBS and DGS methods.

ID	T	P	RT-DGS	Sim-DGS	RT-RBS	Sim-RBS
m_1	12	4	6	3	4	3
m_2	19	7	8	5	6	3
m_3	9	3	5	3	3	2
m_4	25	10	5	2	3	2
m_5	5	1	3	3	2	2
m_6	12	4	3	2	3	1
m_7	7	2	2	2	1	1
m_8	16	6	4	2	3	2
m_9	16	6	5	2	3	2
m_{10}	9	3	4	2	3	1
m_{11}	14	5	4	2	3	2
m_{12}	21	8	6	2	4	2
m_{13}	25	10	10	6	5	4
m_{14}	12	4	6	3	4	2
m_{15}	19	7	8	5	6	3
m_{16}	9	3	5	3	3	2
m_{17}	12	4	6	3	4	2
m_{18}	21	8	5	2	3	2
m_{19}	16	6	7	4	5	3
m_{20}	12	4	6	3	4	3

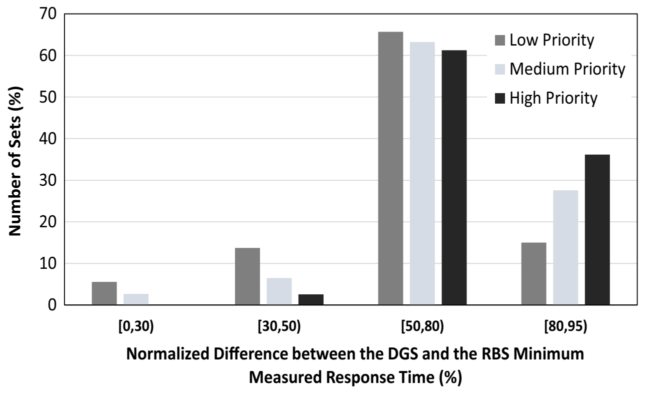


Figure 14 The difference between the minimum measured response times.

Moreover, according to the simulation results, the response times measured in the simulation in the RBS method are lower or equal to the ones measured in the DGS method. The measured response times are equal in the methods for the messages with lower priority, whereas for higher priority messages the RBS method provides faster response time, e.g., m_5 and m_7.

7.3 Simulation on Different Architectures

In this section, we compare the RBS and DGS methods based on the measured response time of the messages in a set of randomly generated network architectures. We generated 50 random network architectures, where the number of hierarchy level is chosen within [2, 4], the number of HaRTES switches in the network is selected within [3, 7], and the number of nodes connected to the switches is selected within [3, 35]. Number of hierarchy level shows the size of the architecture with respect to its depth. For instance, the number of hierarchy level in the network depicted in Fig. 9 is 2, one is the master switch and the other is its children. Also, the number of hierarchy level in the

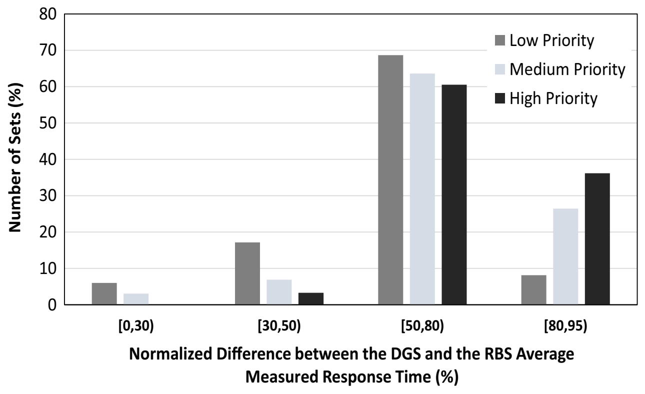


Figure 15 The difference between the average measured response times.

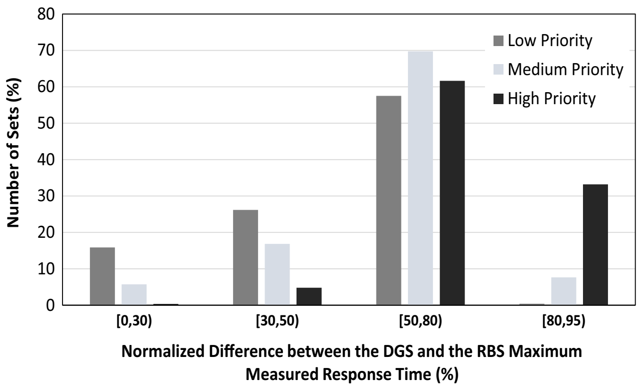


Figure 16 The difference between the maximum measured response times.

network depicted in Fig. 12 is 4. The nodes are randomly attached to the switches such that each switch has at least one node connected to.

Based on the size of the generated network, a set of synchronous global messages is generated. The number of messages in the set is selected within [20, 140]. The messages are assigned to the nodes randomly such that each node in the network is a source of at least one message. The transmission time of the messages is constant and $123\mu s$, i.e., each message contains one packet, only. The periods of the messages is selected within $[5, 15]ECs$ and the deadlines are chosen to be equal to the periods. Moreover, the priority of the messages is set based on the Rate Monotonic algorithm.

Furthermore, the capacity of the links in the generated network is set to 100Mbps. The EC size is selected within $[1, 7]ms$, where between 50 % to 80 % of that is allocated, randomly, to the synchronous window.

We simulated each generated network for 50000 ECs in both RBS and DGS methods. We measured the minimum, average and maximum response time of the messages in both methods. For each experiment (each generated network architecture), we focused on three messages, the highest priority, medium priority, and the lowest priority. We derived the normalized difference of the measured response times in both methods for the three messages, using the expression (11). Figure 14 illustrates the normalized difference of the minimum measured response times of the messages in the RBS and DGS methods. Similar to the comparison illustration in Section 7.1, the x-axis shows the normalized difference within a certain values, from 0 % to 95 %. Also, y-axis shows the number of sets that have the normalized differences within the specific range.

As it can be seen in the figure, all normalized differences for the three messages are positive, which means that in all generated experiments the RBS method delivers the messages faster than the DGS methods. Moreover, among the measured messages, the highest priority message has the biggest difference. This shows that RBS method performs better for high priority messages, as it was concluded in the previous section. For instance, around 33 % of the sets has the normalized difference of minimum measured response times within [80, 95] % for the highest priority message.

Figures 15 and 16 show the normalized differences of the average and maximum response times, respectively. The behavior follows the same way as the one described for Fig. 14. Moreover, the pattern of the figures is similar to the ones depicted in Figs. 10 and 13, that can validate our experiment comparison based on the response time analysis. However, in the simulation, we could not find any negative normalized difference of response times, which means that we could not capture any experiment in which the DGS method performs better than the RBS method.

8 Hardware Changes for Supporting the RBS

Figure 1 depicts the internal structure of the HaRTES switch. Looking more closely to the synchronous part of the output ports, it can be seen that the synchronous messages are held in FIFO queues since the dispatched traffic fits in one EC. However, as presented in this section, in interlink output ports, traffic has no such guarantees and can

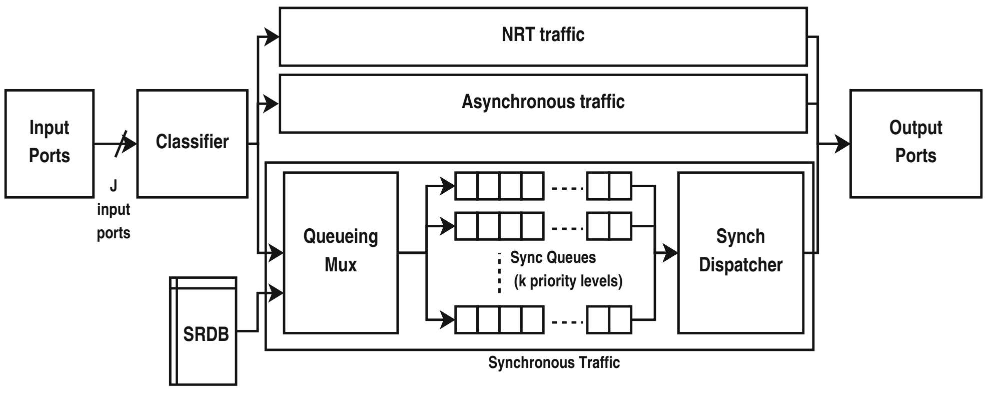
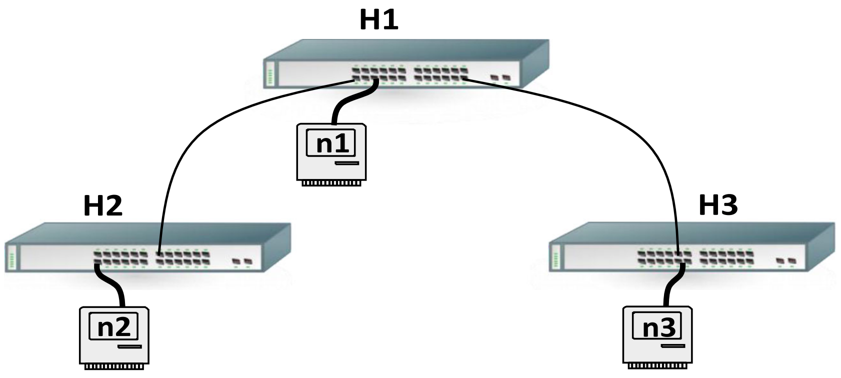


Figure 17 Modified switch architecture.

Figure 18 Experiment setup.



accumulate from EC to EC. In this scenario the use of FIFO queues would lead to a poor real-time performance, due to priority inversions. For this reason, in the RBS method, the output ports use priority queues.

The HaRTES switch is implemented in FPGA technology. To allow wire speed treatment of the incoming messages, all functionalities directly associated with the internal message handling (e.g. classification, policing, forwarding) are implemented in hardware. Priority queues can be trivially implemented in software, but in hardware the case is completely different. There are many alternatives that present different tradeoffs between performance and resource utilization [43].

Several options have been considered and that the one actually implemented presents a balance between several requirements. In particular, the packet processing takes less time than the reception time of a minimum-size Ethernet frame, thus enabling wire speed operation, and the amount of required resources (logic blocks and memory) is limited and configurable, being proportional to the needed priority levels.

Figure 17 shows the internal architecture of the output port management. To make the figure more readable, it is represented only one output port. Moreover, only the synchronous packet management block is detailed. Real-time asynchronous packets are handled in the same way as the synchronous ones and NRT traffic is put in a single FIFO queue. Firstly, as in all ports, there is a packet classifier that inspects the incoming packets and determines its class (synchronous, asynchronous and non real-time). Then, real-time traffic (both synchronous and asynchronous) is processed by a block called "Queuing Mux". This block checks the priority of each packet, stored in the System Requirements Database (SRDB) and places it in the appropriate queue (i.e., the one that corresponds to the message priority). Note that messages with different priorities are placed in different queues, thus each individual queue is handled in FIFO order. As for normal ports, there is also a dispatcher. However, in addition to check for the current EC phase, the dispatcher also has to process each one of the queues according to its priority. To achieve this goal, the dispatcher processes the queues in a given predefined order, which corresponds to the queue priorities. The dispatcher only moves to a different queue when all the higher priority ones are empty.

The amount of resources depends on the number of messages and priorities. In particular, each implemented priority level requires the static allocation of a portion of memory sufficiently high to hold the (eventually) several pointers to messages waiting in the queue. Thus, the total amount of memory may easily become unbearable. For this reason it was decided to associate explicitly each queue with a specific priority level. This scheme allows creating only the needed amount of queues, thus saving resources.

9 Experimental Validation

For this evaluation we defined a network consisting of three switches and three nodes, connected in a topology shown in Fig. 18.

Table 4 The messages parameters for the prototype experiment.

ID	T	P	S	Ds	ID	T	P	S	Ds
m_1	20	6	3	1	m_{16}	18	5	3	2
m_2	20	6	1	3	m_{17}	15	4	2	3
m_3	25	7	3	1	m_{18}	15	4	3	2
m_4	15	4	3	2	m_{19}	20	6	3	2
m_5	10	2	3	2	m_{20}	10	2	2	3
m_6	15	4	3	2	m_{21}	18	5	3	2
m_7	10	2	2	1	m_{22}	25	7	3	2
m_8	20	6	3	2	m_{23}	18	5	3	2
m_9	12	3	2	3	m_{24}	5	1	3	1
m_{10}	5	1	3	1	m_{25}	15	4	2	3
m_{11}	15	4	2	1	m_{26}	15	4	2	3
m_{12}	10	2	3	1	m_{27}	18	5	1	2
m_{13}	15	4	3	2	m_{28}	10	2	1	2
m_{14}	18	5	3	2	m_{29}	10	2	2	1
m_{15}	25	7	3	1	m_{30}	10	2	3	2

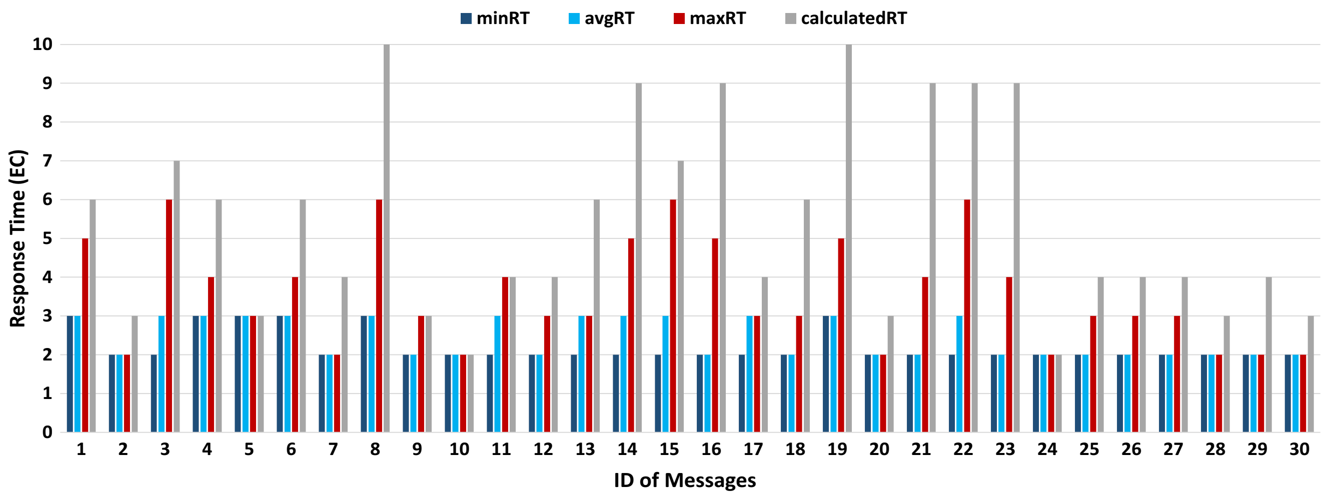


Figure 19 The minimum, average, maximum measured and computed response times.

In this experiment the capacity of the links is 100Mbps. Currently, the HaRTES switch implementation uses netF-PGA boards with four ports per switch, that are used for inter-links and local-links. According to the extensive measurements reported in [44], the fabric latency of the HaRTES switch varies between $2\mu s$ and $2.4\mu s$, with an average of $2.2\mu s$.

We set the EC size to $1ms$ and we allocated $700\mu s$ for the synchronous window. Also, we generated 30 synchronous global messages with only one packet and $123\mu s$ transmission time. The message periods are uniformly selected within $[5, 25]ECs$ and the priority is assigned based on Rate Monotonic policy. Table 4 shows the message parameters including the period (T), priority (P), source node (S) and destination node (Ds).

We measured the response time of the messages during 60000 ECs. In Fig. 19, we show the minimum, average and maximum values of the measured response times (minRT, avgRT, maxRT). Note that the measured response times are expressed in number of EC, thus, due to this coarse resolution, it may happen that the average and the maximum are equal. Moreover, we calculated the response time of the messages using the presented analysis in this paper. The computed response times (calculatedRT) are also illustrated in Fig. 19 to show the difference of the measured and computed values.

Analyzing Fig. 19, we can see that the measured response times are lower or equal than the response time predicted by the analysis. It can also be observed that the analysis embodies a certain degree of pessimism, as discussed in Section 6. As expected, the degree of pessimism depends on the message priority and number of hops. For instance, the difference between the predicted and observed response times for messages with priority 1 and 2 is at most 1 ECs, while for messages with priority 6 and 7 this difference varies between 2 and 5 ECs.

We believe that the pessimism in the analysis stems from several sources. We discuss these sources herein, however a detail study of the pessimism and the solutions for them are not discussed in this paper and remained as future work.

The first source of the pessimism is related to the phasing of the messages. According to Eq. 8, all higher and equal priority messages that share links with the message under analysis are taken into account for calculation. However, due to different phasing the messages may not interfere in the shared link. Assume the network example depicted in Fig. 18. Also, assume that two messages, m_1 from node n2 to n3 and m_2 from node n1 to n3, have share link in the interlink between switch H1 and switch H3. Although they have share links, in practice, m_2 can be received by switch H3 while m_1 is buffered in switch H1, hence they do not interfere.

The other source is related to the idle time calculation. We considered the largest packet to compute the idle time in Eq. 5. However, the idle time may occur for smaller packets in reality.

10 Conclusion and Future Work

This work addressed the problem of supporting real-time multi-hop traffic in HaRTES architectures. Recently, we developed two forwarding methods, namely DGS, based on buffering, and RBS, based on immediate forwarding. In this paper we establish the superiority of RBS, which allows a significant reduction of the end-to-end response times and we showed how it can be efficiently implemented within HaRTES.

Concerning the RBS versus DGS comparison, we first compared their performance applying the response time analysis of each method on two different networks. We have shown that the response time of the highest priority messages is always smaller using the RBS method and the response time of the lowest and medium priority messages is equal or, in most of cases, smaller than when applying the DGS method. We have also shown that this difference increases for larger networks. Secondly, we modified a simulation tool to support the RBS method. We compared the response time of a set of messages in the RBS and the DGS methods by means of simulation. Moreover, we generated random network architectures with random set of messages to validate our conclusion from the response time comparison in different network architectures. Finally, we experimentally validated the RBS method with an actual prototype implementation. Our future work aims at removing some pessimism and we also aim at using this real-time multi-hop framework to provide end-to-end resource reservation in large Ethernet networks.

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

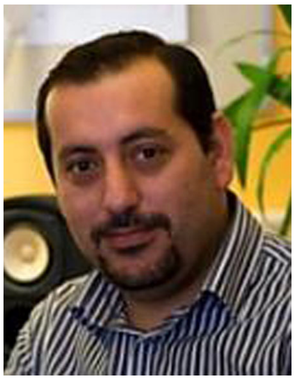
References

1. Decotignie, J.-D. (2005). Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, *93*(6), 1102–1117.
2. Steiner, W., Bauer, G., Hall, B., Paulitsch, M., & Varadarajan, S. (2009). TTEthernet dataflow concept. In *8th IEEE International Symposium on Network Computing and Applications*.
3. Hanzalek, Z., Burget, P., & Sucha, P. (2009). Profinet IO IRT message scheduling. In *21st Euromicro Conf. on Real-Time Sys. (ECRTS)*.
4. IEEE (2011). IEEE Std. 802.1as-2011, ieee standard for local and metropolitan area networks-timing and synchronization for time-sensitive applications in bridged local area networks. *IEEE, Technical Report*.
5. IEEE (2011). IEEE Std. 802.1qat, ieee standard for local and metropolitan area networks, virtual bridged local area networks, amendment 14: Stream reservation protocol. *IEEE, Technical Report*.
6. IEEE (2011). IEEE Std. 802.1qav, ieee standard for local and metropolitan area networks, virtual bridged local areanetworks, amendment 12: Forwarding and queuing enhancements for time-sensitive streams. *IEEE, Technical Report*.
7. Gomez-Molinero, F. (July 2007). Real-time requirement of media control applications. In *19th Euromicro Conference on Real-Time Systems (ECRTS)*.
8. Cho, C.-S., Chung, B.-M., & Park, M.-J. (2005). Development of real-time vision-based fabric inspection system. *IEEE Transactions on Industrial Electronics*, *52*(4), 1073–1079.
9. Kumar, A. (2008). Computer-vision-based fabric defect detection: A survey. *IEEE Transactions on Industrial Electronics*, *55*(1), 348–363.
10. Hwang, C.-L., & Shih, C.-Y. (2009). A distributed active-vision network-space approach for the navigation of a car-like wheeled robot. *IEEE Transactions on Industrial Electronics*, *56*(3), 846–855.
11. Lim, H.-T., Volker, L., & Herrscher, D. (2011). Challenges in a future IP/Ethernet-based in-car network for real-time applications. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*.
12. Lim, H.-T., Weckemann, K., & Herrscher, D. (2011). Performance study of an in-car switched ethernet network without prioritization. In *Proceedings of the Third international conference on Communication technologies for vehicles*. Springer.
13. Santos, R., Behnam, M., Nolte, T., Pedreiras, P., & Almeida, L. (2011). Multi-level hierarchical scheduling in ethernet switches. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*.
14. Ashjaei, M., Pedreiras, P., Behnam, M., Bril, R.J., Almeida, L., & Nolte, T. (2014). Response time analysis of multi-hop HaRTES ethernet switch networks. In *9th International Workshop on Factory Communication Systems (WFCS)*.
15. Ashjaei, M., Behnam, M., Pedreiras, P., Bril, R.J., Almeida, L., & Nolte, T. (2014). Reduced buffering solution for multi-hop HaRTES switched Ethernet networks. In *The 20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications (RTCSA)*.
16. Pedreiras, P., & Almeida, L. (2005). The Industrial Communication Systems Handbook. CRC Press, ch. Approaches to Enforce Real-Time Behavior in Ethernet, ISBN: 0-8493-3077-7.
17. Varadarajan, S., & Chiueh, T. (1998). EtheReal: a host-transparent real-time fast ethernet switch. In *6th International Conference on Network Protocols*.
18. Hoang, H., & Jonsson, M. (2003). Switched real-time ethernet in industrial applications - deadline partitioning. In *9th Asia-Pacific Conference on Communications (APCC)*.
19. (2013). EPSG Draft Standard 301 Ethernet POWERLINK Communication Profile Specification Version 1.2.0, Ethernet POWERLINK Standardisation Group.
20. (2010). IEC 61158, industrial communication networks - Fieldbus specifications.
21. Carvajal, G., Figueroa, M., Trausmuth, R., & Fischmeister, S. (2013). Atacama: An open FPGA-Based platform for mixed-criticality communication in multi-segmented Ethernet networks. In *21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.
22. Alderisi, G., Patti, G., & Bello, L. (2013). Introducing support for scheduled traffic over IEEE audio video bridging networks. In *18th IEEE Conference on Emerging Technologies Factory Automation (ETFA)*.
23. Time-sensitive networking task group, available at http://www.ieee802.org/1/pages/tsn.html.
24. Marau, R., Almeida, L., & Pedreiras, P. (2006). Enhancing real-time communication over COTS Ethernet switches. In *6th IEEE International Workshop on Factory Communication Systems (WFCS)*.
25. Santos, R., Vieira, A., Pedreiras, P., Oliveira, A., Almeida, L., Marau, R., & Nolte, T. (2010). Flexible, efficient and robust real-time communication with server-based Ethernet switching. In *8th IEEE International Workshop on Factory Communication Systems (WFCS)*.
26. Ashjaei, M., Behnam, M., Almeida, L., & Nolte, T. (2013). Performance analysis of master-slave multi-hop switched ethernet networks. In *8th IEEE Int. Symp. on Industrial Embedded Systems (SIES)*.
27. Mifdaoui, A., Frances, F., & Fraboul, C. (2010). Performance analysis of a master/slave switched ethernet for military embedded

applications. *IEEE Transactions on Industrial Informatics*, *6*(4), 534–547.
28. Zhang, M., Shi, J., Zhang, T., & Hu, Y. (2008). Hard real-time communication over multi-hop switched ethernet. In *The IEEE Int. Conference on Networking, Architecture, and Storage (NAS)*.
29. Charara, H., Scharbarg, J.-L., Ermont, J., & Fraboul, C. (2006). Methods for bounding end-to-end delays on an AFDX network. In *18th Euromicro Conference on Real-Time Systems(ECRTS)*.
30. Bauer, H., Scharbarg, J.-L., & Fraboul, C. (2010). *Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach.* IEEE Transaction on Industrial Informatics.
31. Kemayo, G., Ridouard, F., Bauer, H., & Richard, P. (2013). Optimistic problems in the trajectory approach in fifo context. In *18th IEEE Conf. on Emerging Technologies Factory Automation (ETFA)*.
32. Li, X., Cros, O., & George, L. (2014). The trajectory approach for AFDX FIFO networks revisited and corrected. In *The 20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications (RTCSA)*.
33. Queck, R. (2012). Analysis of Ethernet AVB for automotive networks using network calculus. In *IEEE International Conference on Vehicular Electronics and Safety (ICVES)*.
34. Manderscheid, M., & Langer, F. (2011). Network calculus for the validation of automotive ethernet in-vehicle network configurations. In *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*.
35. Bordoloi, U.D., Aminifar, A., Eles, P., & Peng, Z. (2014). Schedulability analysis of ethernet AVB switches. In *The 20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications (RTCSA)*.
36. Lenzini, L., Martorini, L., Mingozzi, E., & Stea, G. (2006). Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks. *Elsevier Performance Evaluation*, vol. 63.
37. Schmitt, J., Zdarsky, F., & Fidler, M. (2008). Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch... In *The 27th IEEE Conference on Computer Communications*.
38. Lenzini, L., Martorini, L., Mingozzi, E., & Stea, G. (2006). 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*.
39. Ashjaei, M., Behnam, M., Rodriguez-Navas, G., & Nolte, T. (2013). Implementing a clock synchronization protocol on a multi-master switched ethernet network. In *18th Conference on Emerging Technologies Factory Automation (ETFA)*.
40. Gessner, D., Proenza, J., Barranco, M., & Portugal, P. (2014). Towards a reliability analysis of the design space for the communication subsystem of ft4ftt. In *19th IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*.
41. Gessner, D., Proenza, J., & Barranco, M. (2014). A proposal for managing the redundancy provided by the flexible time-triggered replicated star for ethernet. In *10th IEEE Workshop on Factory Communication Systems (WFCS)*.
42. Ashjaei, M., Behnam, M., & Nolte, T. (2012). The design and implementation of a simulator for switched ethernet networks. In *3rd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*.
43. Huang, M., Lim, K., & Cong, J. (2014). A scalable, high-performance customized priority queue. In *24th International Conference on Field Programmable Logic and Applications*.
44. Santos, R. (2010). *Enhanced Ethernet Switching Technology for Adaptive Hard Real-Time Applications*. PhD Thesis, University of Aveiro, Aveiro, Portugal.

Mohammad Ashjaei is a PhD student at Mälardalen University since April 2012. He studied Electrical Engineering in Tehan, Iran and received his B.Sc. in 2003. He has been working for many private companies as a hardware programmer. Then, Mohammad moved to Sweden in 2010 and studied computer science with emphasize on real-time embedded systems at Mälardalen University. He received his M.Sc. degree in 2012 and in the same year he started his PhD studies in the same University. Mohammad was a visiting researcher at University of Aveiro, Portugal, for one month in 2013. His main research interests are real-time distributed systems, response time analysis, modeling and development of related algorithms.

Luis Silva was born in Portugal on July 25, 1987. He received the M.Sc. degree in electronics and telecommunications engineering from the University of Aveiro, Portugal, in 2010. He is currently a Ph.D. student on the 2013/2014 MAP-Tele doctoral program in telecommunications, a joint venture of three Portuguese universities: Minho, Aveiro and Oporto. He is also a researcher at the Institute for Telecommunications, Aveiro, Portugal. His main research interests include distributed embedded systems, real-time networks and cooperative intelligent transportation systems.



Moris Behnam has awarded a B.Eng., and M.Sc. in Computer and Control Engineering at the University of Technology, Iraq, and also MS.c., Licentiate, and PhD in Computer Science and Engineering at MDH, Sweden, in 1995, 1998, 2005, 2008 and 2010 respectively. Moris has been a visiting researcher at Wayne State University, USA in 2009 and he has been a Postdoctoral Researcher at University of Porto in 2011. His research interests include real-time scheduling, synchronization protocols, multicore/multiprocessor systems, distributed embedded real-time systems and using control theories in real-time scheduling.

Paulo Pedreiras was born in Aveiro, Portugal, in 1967. He graduated in Electronics and Telecommunications Engineering, in 1997, and received the Ph.D. degree in Electrotechnical Engineering, in 2003, at the University of Aveiro, Portugal. He currently is Assistant Professor at the Electronics, Telecommunications and Informatics Department of the University of Aveiro, Portugal. He is also affiliated with the Portuguese Telecommunications Institute, Aveiro site. He has been involved, as a researcher, in 12 national and European research projects and has been the principal investigator of two national projects. His current research interests include real-time systems, networked embedded systems, real-time communication protocols, industrial communications, wireless communications, intelligent transportation systems and robotics. Since 2000 he has authored or co-authored more than 100 papers in international peer-reviewed journals and conferences. He participates regularly on the technical program committees of some of the major events of his research area, such as WFCS, SIES and ETFA. He collaborates regularly, as reviewer, in several top international journals such as IEEE Transactions on Industrial Informatics, IEEE Transactions on Industrial Electronics, Journal of Systems Architecture (Elsevier) and Springer Real-Time Systems Journal. He has been the Program Co-Chair of APRES 2012, WiP Co-chair of WFCS 2012, PC Co-chair of WFCS 2015 and member of the local organizing committee of several national scientific events.

Luis Almeida graduated in Electronics and Telecommunications Eng. in 1988 and received a Ph.D. in Electrical Eng. in 1999, both from the University of Aveiro in Portugal. He is currently an associate professor in the Electrical and Computer Engineering Department of the University of Porto (UP), Portugal, and a senior researcher in the Telecommunications Institute at UP where he coordinates the Distributed and Real-Time Embedded Systems (DaRTES) lab. Among several appointments, he was a member of the IEEE Technical Committee on Real-Time Systems (2008-2013), Program and General Chair of the IEEE Real-Time Systems Symposium (2011-2012 respectively) and Vice-President of the RoboCup Federation (2011-2013) being Trustee of this organization since 2008. His research interests include those related to real-time networks for distributed industrial/embedded systems including for teams of mobile robots.

Reinder J. Bril received a B.Sc. and a M.Sc. (both with honors) from the University of Twente, and a Ph.D. from the Technische Universiteit Eindhoven, the Netherlands. He started his professional career in January 1984 at the Delft University of Technology. From May 1985 till August 2004, he has been with Philips, and worked in both Philips Research as well as Philips' Business Units. He worked on various topics, including fault tolerance, formal specifications, software architecture analysis, and dynamic resource management, and in different application domains, e.g. high-volume electronics consumer products and (low volume) professional systems. In September 2004, he made a transfer back to the academic world, i.e. to the System Architecture and Networking (SAN) group of the Mathematics and Computer Science department of the Technische Universiteit Eindhoven. His main research interests are currently in the area of reservationbased resource management for networked embedded systems with real-time constraints.

Thomas Nolte was awarded a B.Eng., M.Sc., Licentiate, and Ph.D. degree in Computer Engineering from Mälardalen University (MDH), Västerås, Sweden, in 2001, 2002, 2003, and 2006, respectively. He has been a Visiting Researcher at University of California, Irvine (UCI), Los Angeles, USA, in 2002, and a Visiting Researcher at University of Catania, Italy, in 2005. He has been a Postdoctoral Researcher at University of Catania in 2006, and at MDH in 2006-2007. Thomas Nolte became an Assistant Professor at MDH in 2008, and Associate Professor at MDH in 2009. 2012 he became Full Professor of Computer Science.