Mälardalen University Press Dissertations
No. 211

REAL-TIME COMMUNICATION OVER SWITCHED ETHERNET WITH RESOURCE RESERVATION

Mohammad Ashjaei

2016

School of Innovation, Design and Engineering

ISBN 978-91-7485-285-1
ISSN 1651-4238
Printed by Arkitektkopia, Västerås, Sweden

Mälardalen University Press Dissertations
No. 211

REAL-TIME COMMUNICATION OVER SWITCHED ETHERNET WITH RESOURCE RESERVATION

Mohammad Ashjaei

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras torsdagen den 10 november 2016, 13.15 i Lambda, Mälardalens högskola, Västerås.

Fakultetsopponent: Associate Professor Thilo Sauter, Vienna University of Technology, Austria

Akademin för innovation, design och teknik

Abstract

Due to the need for advanced computer-controlled functionality in distributed embedded systems the requirements on network communication are becoming overly intricate. This dissertation targets the requirements that are concerned with real-time guarantees, run-time adaptation, resource utilization and flexibility during the development. The Flexible Time-Triggered Switched Ethernet (FTT-SE) and Hard Real-Time Ethernet Switching (HaRTES) network architectures have emerged as two promising solutions that can cater for these requirements. However, these architectures do not support multi-hop communication as they are originally developed for single-switch networks. This dissertation presents a fundamental contribution in multi-hop real-time communication over the FTT-SE and HaRTES architectures targeting the above mentioned requirements. It proposes and evaluates various solutions for scheduling and forwarding the traffic through multiple switches in these architectures. These solutions preserve the ability of dynamic adaptation without jeopardizing real-time properties of the architectures. Moreover, the dissertation presents schedulability analyses for the timeliness verification and evaluation of the proposed solutions as well as several protocols to support run-time adaptation in the multi-hop communication. Finally, the work led to an end-to-end resource reservation framework, based on the proposed multi-hop architectures, to support flexibility during the development of the systems. The efficiency of the proposed solutions is evaluated on various case studies that are inspired from industrial systems.

ISBN 978-91-7485-285-1
ISSN 1651-4238

Populärvetenskaplig sammanfattning

Användandet av datorer för att underlätta vardagen i våra liv visar på en stor ökning av antalet datorer under de senaste decennierna. Datorer har många olika användningsområden för att klara av enklare vardagliga saker. Datorer finns i allt från mobiltelefoner och hushållsapparater till maskiner som utför mer komplicerade och sofistikerade uppgifter. Dessa maskiner återfinns i avancerade system så som moderna bilar och inom automation av industri och tillverkning. I motsats till klassiska datorer med tangentbord och skärm, utformade till att kunna göra en uppsjö av olika uppgifter, är dessa datorsystem konstruerade för att utföra specifika funktioner. Vi kallar dessa datorsystem inbyggda system, där datorn är inbyggd i systemet och utför en mer eller mindre specifik uppgift. För att få en funktionalitet med hjälp av dessa specifika system är själva funktionen ofta fördelad över många olika inbyggda system. De inbyggda systemen som ingår in den specifika funktionen är förbundna med varandra via ett kommunikationssystem. Dessa system kallar vi distribuerade inbyggda system. Ett exempel på ett distribuerat inbyggt system är en krockkudde i en modern bil. Systemet innehåller flera sensorer och lämnar information till en styrenhet för att besluta om bilen har krockat. Om bilen har krockat ska styrenheten bestämma hur och när krockkudden ska blåsas upp. Styrenheten har i sin tur sensorer som skickar meddelanden i det distribuerade inbyggda systemet.

Många distribuerade inbyggda system har specifika tidskrav som måste uppfyllas för att korrekt funktionalitet ska uppnås. Meddelanden som skickas över nätverket måste tas emot av mottagare inom en viss tid som kallas deadline. Denna typ av system kallas distribuerade inbyggda realtidssystem. Beroende på vilken typ av funktionalitet som används, så kan en miss av deadline leda till katastrofala konsekvenser. Till exempel så kan ett sent meddelande gällande

krock av bilen leda till att krockkudden blåses upp för sent och därmed motverkar sitt syfte. För att säkerställa tidsmässig leverans av meddelanden så kan ett protokoll användas för att skicka och ta emot meddelanden på ett förutsägbart sätt. Detta protokoll används sedan av alla inbyggda system sammankopplade med kommunikationssystemet, vilket leder till att systemen kan prata med varandra och förstå varandra.

I denna avhandling fokuserar vi särskilt på två protokoll som är baserade på Ethernet-teknik. Dessa protokoll är FTT-SE och HaRTES, som ursprungligen är utvecklade för relativt enkla nätverk bestående av en enda Ethernet-switch. I många industriella tillämpningar finns tiotals inbyggda system som i och med sitt stora antal inte kan sammankopplas av endast en Ethernet-switch. Därför behövs en lösning för kommunikationssystem där flera Ethernet-switchar är sammankopplade, samt att garantier kan ges för den tidsmässiga leveransen av meddelanden som skickas över flera switchar. Det huvudsakliga målet med denna avhandling är att föreslå lösningar för att anpassa FTT-SE och HaRTES för distribuerade inbyggda realtidssystem bestående av flera switchar. Vi presenterar tidsanalyser för alla föreslagna lösningarna så att tidsgarantier kan ges. Vidare utvärderar vi våra lösningar med hjälp av simulering så att de kan jämföras samt så att vi kan vet vad som gäller for och presentera en effektiv lösning för varje typ av system.

Abstract

Due to the need for advanced computer-controlled functionality in distributed embedded systems the requirements on network communication are becoming overly intricate. This dissertation targets the requirements that are concerned with real-time guarantees, run-time adaptation, resource utilization and flexibility during the development of such systems. The Flexible Time-Triggered Switched Ethernet (FTT-SE) and Hard Real-Time Ethernet Switching (HaRTES) network architectures have emerged as two promising solutions that can cater for these requirements. However, these architectures do not support multi-hop communication as they are originally developed for single-switch networks. This dissertation presents a fundamental contribution in multi-hop real-time communication over the FTT-SE and HaRTES architectures targeting the above mentioned requirements. It proposes and evaluates various solutions for scheduling and forwarding the traffic through multiple switches in these architectures. These solutions preserve the ability of dynamic adaptation without jeopardizing real-time properties of the architectures. Moreover, the dissertation presents schedulability analyses for verification of timeliness and evaluation of the proposed solutions along with several protocols that support run-time adaptation in the context of multi-hop communication. Finally, the work led to an end-to-end resource reservation framework, based on the proposed multi-hop architectures, to support flexibility during the development of the systems. The efficiency of the proposed solutions is evaluated using various case studies that are inspired by industrial systems.

To my lovely girls, Sara and Liana!

Acknowledgments

This section is an attempt to thank those who influenced and assisted me during these years to reach this point of my life, writing my doctoral dissertation. During these years I met fantastic people to work with, I made fabulous friends and I worked in a wonderful environment.

I take this opportunity to express my profound gratitude to my supervisors Prof. Thomas Nolte, Dr. Moris Behnam and Prof. Luis Almeida for their constant encouragement throughout my studies. They have been guiding me from my master thesis with an exemplary support, useful suggestions, comments and feedback. Thomas has always influenced me by his great positive attitude bringing to work. I practically learned from him to think positive in every situation of life. I enjoyed every moment of our discussions, trips and meetings. I also appreciate the valuable feedback and assists of Moris which have improved my work significantly. No matter when, he has always dedicated time for our discussions. I learned from him to think critically without being biased about my work, like a ”third reviewer”. I am very grateful to know and work with Luis, who is full of energy not only in his research life. I learned many fundamental concepts in the research area from him. We had lots of fun activities and discussions during our meetings, trips and visits. It may sound cliché, but without your support this dissertation would not be possible. The person who is writing this acknowledgment has a significantly different view from the one 5 years ago and this is a big achievement for me.

Next, I would like to express a deep sense of gratitude to my co-authors. I am very thankful to Prof. Paulo Pedreiras and Prof. Reinder J. Bril for their cordial support and valuable information. I would also like to thank Dr. Saad Mubeen and Luis Silva for their precious information during the meetings and discussions.

During my studies I had a chance to visit University of Porto and University of Aveiro in Portugal as a visiting researcher for one month. I had a great time

there in a very beautiful September. I gained several nice experiences, visiting small and big cities, tasting delicious cookies and Port wines. I got to know many colleagues working on different areas. I had a lot of great chats and discussions with them. In particular, I would like to thank Paulo Pedreiras and his family, Luis Almeida, Ricardo Marau, Luis Silva and Milton Cunguara. You made my visit full of joy and work, Obrigado!

I am obliged to my friends and colleagues for all the fun we had during these years in conference trips, ”fika” times, parties, and badminton sessions. You must know that you are fabulous and supportive friends. I never felt alone in this city. I also would like to appreciate IDT administration staff for their help with practical issues. The cover artwork is done by my lovely cousin Sara Shekarchi for which I appreciate her help.

Last but not least, I would like to take this opportunity to thank my family. In particular, my parents for their endless love, tremendous support and encouragement during my entire life. I am distinctly thankful to my wife Sara, who was always patient and supportive during all ups and downs in our life. I am a very lucky man to have you. I am blessed to have my little daughter, Liana, in my life. She brought tremendous love to our life. Moreover, I am grateful of my little siblings, Saeedeh and Soheil, whom together we could laugh for few hours constantly!

This work has been supported by the Swedish Foundation for Strategic Research under the project PRESS - Predictable Embedded Software Systems.

Mohammad Ashjaei
October, 2016
Västerås, Sweden

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In modern societies, embedded systems are playing a vital role. In fact they are inevitably becoming part of our daily lives, ranging from tiny electronic devices to more sophisticated systems, such as aircrafts and modern cars. Normally a small electronic device contains only one processing unit that performs a particular function. However, this is not the case in large industrial settings. In many application domains, for instance in automotive, automation and process controls, a functionality is developed over several embedded systems that are communicating via a network media. This system is typically known as a *distributed embedded system*. Often, communication within the distributed embedded system has constraints with respect to timing, where the information should be delivered within a priori defined time. This type of communication is known as *real-time communication*. Besides the timing aspect of real-time communication, other requirements are nowadays imposed due to the need for advanced computer-controlled functionalities. The requirements that are concerned in this dissertation include real-time guarantees, run-time adaptation, resource utilization and flexibility during the development. Considering the mentioned requirements, this dissertation presents a fundamental contribution on multi-hop real-time communication. The first chapter explains the motivation and trend of the dissertation. Then, it states the thesis statement and research challenges, which are followed by presenting an overview of research contributions. Finally, the chapter describes the research methodology and outline of the dissertation.

1.1 Motivation

Over the last decades, it has been witnessed that functionalities given to distributed embedded systems are becoming more intricate with high computing resources involved. This complication is not only due to new complex functions, but also imposed by other complementary requirements. In addition to the timing aspects of real-time communication within the distributed embedded systems, the demand for exchanging a significant amount of heterogeneous data through networks is growing. For instance, in media control systems [1], machine vision [2], and vehicle guidance [3], where real-time audio and video streams are transmitted, the data size can reach to hundreds of KBytes per transaction. As a result, the network should support high transmission bandwidth. Moreover, the current design trend aims at building a system in a composable way, which means designing different parts separately and integrating them in a way that maintains the properties of each part. Therefore, there is a demand for *resource reservation* as one of the prominent techniques in utilizing available resources effectively for composability purposes. According to this technique, a fraction of available resources is dedicated to a subset of the system, hence the subset can be developed independently. The latter brings flexibility during the development of the system. In this scope, the system can be designed to allow network resources for the data transmission to be traded online for Quality of Service (QoS) within acceptable ranges. In this context, QoS may refer to throughput, transmission delay, jitter and other performance metrics. This means that if there is a request for higher QoS for a subset of the system, the network bandwidth allocated for the subset can be increased during run-time of the system as long as there is available total bandwidth or the bandwidth of other subsystems can be reduced. Besides, in some applications high availability and adaptivity are particularly important to maintain the system usefulness, which is incompatible with restarting the system or taking it offline to carry out a reconfiguration. Therefore, the parameters of the system should be changed during run-time. It should be noted that timeliness behavior of the system during the changes must be maintained. *Dynamic reconfiguration* techniques are used to obtain the above mentioned requirements.

In order to deal with such complexity and requirements, solutions based on switched Ethernet have been proposed. The reasons can be found in the features of Ethernet, including its high bandwidth support, being a well established technology and being easy to deploy. Compared to other network technologies used in embedded real-time systems, Ethernet can support a higher

throughput[1]. Moreover, it is expandable and flexible in the sense that switches can be connected together in different topologies to build a *multi-hop* architecture. Due to the wide availability of Ethernet switches, using them in industrial applications is also cost-effective.

In the context of switched Ethernet, many academic and industrial solutions have been actively developed, where the prominent ones are Time-Triggered Ethernet [4], PROFINET IRT [5], Ethernet POWERLINK [6], EtherCAT [7] and Atacama [8]. All these protocols are based on the time-triggered communication paradigm, complemented with some support for event-triggered communication. In time-triggered communication, transmission of traffic is done according to a predefined time schedule. On the other hand, in event-triggered communication, transmission of traffic is done based on occurrence of an event during run-time. Time-triggered traffic is statically configured at pre-runtime and cannot be changed online. At best there is support to a predefined set of modes that can be selected online. Therefore, the protocols that are optimized for static scheduling are not suitable to handle dynamic real-time applications, where changes in system parameters may be requested online. Avionics Full DupleX Switched Ethernet (AFDX) [9] is a standard that defines the protocol specifications for the data exchange between avionics subsystems. It is based on standard Ethernet switching components that performs the standard functions, such as filtering, policing, and forwarding. However, the traffic is statically configured to assure a deterministic behavior. More recently, Audio Video Bridging (AVB), a set of technical IEEE standards [10, 11, 12], is gaining momentum, mainly among the automotive industries. Ethernet AVB supports IEEE 1588-based clock synchronization, bandwidth reservations and traffic shaping services, at switches and end nodes. Recently, an effort from academia to add offline scheduled services to AVB has been done [13]. The IEEE 802.1 Time Sensitive Networking (TSN) Task Group [14] is continuing the work previously developed in the scope of 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. Overall, the strategies that have been followed by the mentioned switched Ethernet solutions typically favor a certain aspect. For instance, some support static time-triggered services, thus they do not provide dynamic reconfiguration, while other cater for dynamic requirements using event-triggered services but show a poor performance due to lack of an admission control.

[1] Using 1Gbps in industrial applications is becoming more common but the most disseminated version is still Fast Ethernet at 100Mbps.

The above observation led originally to the development of the Flexible Time-Triggered Switched Ethernet (FTT-SE) protocol [15] and the Hard Real-Time Ethernet Switching (HaRTES) architecture [16]. Both architectures are based on the master-slave technique where a master module coordinates the traffic transmission within fixed-duration time-slots. Therefore, it prevents burstiness on the switch ports. The architectures support time-triggered and event-triggered traffic types, provide QoS management and admission control capable of performing run-time adaptation, and flexibility in reserving resources for each traffic type. The transmission bandwidth is reserved for time-triggered and event-triggered traffic separately, which in turn yields a temporal isolation between them. Therefore, the FTT-SE and HaRTES architectures have emerged as two promising solutions that cater for the mentioned requirements. 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 architectures. In particular, the master module in the FTT-SE architecture is developed in a particular node connected to the switch and the architecture uses Commercial Off-The-Shelf (COTS) switches. Whereas, in the HaRTES architecture the master module is inserted inside the HaRTES switch. Moreover, the QoS management and the admission control are developed inside the HaRTES switch. Detailed descriptions of the mentioned network architectures is provided in chapters 2 and 3.

1.2 Thesis and research questions

The FTT-SE and HaRTES architectures provide real-time communication services based on the Flexible Time-Triggered (FTT) paradigm [17]. Nevertheless, these architectures do not support multi-hop communication as they are originally developed for single-switch networks. The distributed embedded systems in industrial applications usually comprise a high number of nodes, which is far beyond the capability of a single switch. Therefore, multi-hop communication is necessary for such applications. In this scope, the thesis is stated as follows:

> *The FTT-SE and HaRTES architectures can provide multi-hop communication without jeopardizing real-time properties, enabling end-to-end adaptable real-time communication with resource reservations.*

According to the thesis statement, there are three main aspects to consider for the mentioned network architectures, which are: (i) providing multi-hop

communication over the FTT-SE and HaRTES networks, (ii) preserving the features of the single-switch network, such as support for time-triggered and event-triggered traffic and resource reservation for different traffic types, and (iii) possibility to extend the solutions to provide end-to-end guarantees resulting in a framework for distributed embedded systems considering both computation and communication resources. Focusing on the first and second aspects, provided solutions should offer high performance with respect to the resource usage. Therefore, the first research question is formulated as follows:

Research Question 1 (RQ1): Which solutions offer better performance in terms of resource usage when extending the FTT-SE and HaRTES architectures to multi-hop communication?

One of the key properties of the architectures is the interaction with dynamic environments by means of dynamic reconfiguration in the networks. This brings to the second research question, that is formulated as follows:

Research Question 2 (RQ2): Which dynamic reconfiguration protocols can be used in the FTT-SE and HaRTES networks and at the same time be compatible with their traffic scheduling and forwarding algorithms?

Once the multi-hop communication considering resource reservation for different traffic types is achieved, the bigger picture is to merge resource reservation within computational nodes and communication in order to present an end-to-end reservation framework. This is the third aspect of the thesis statement. Therefore, the last research question is formulated as follows:

Research Question 3 (RQ3): How can we extend the results from the multi-hop communication to obtain an end-to-end resource reservation framework, including computational and communication resources?

1.3 Overview of contributions

In this dissertation we present four main contributions that answer the formulated research questions. These contributions are presented in this section.

1.3.1 Multi-hop communication in the FTT-SE architecture

This contribution addresses the FTT-SE architecture. The architecture is specified in the context of single-switch networks and the master functionality is developed in a specific node connected to the switch. Connecting the switches together in order to build a multi-hop communication raises many challenges for the FTT-SE architecture, which in turn makes the existing protocol insufficient to handle the traffic scheduling. The scheduler in the master node is required to take into account all switching delays and interference that may occur to a message crossing the switches in order to schedule the message. Moreover, the scheduler needs to have an overall view on the message transmission in all links between the switches and the nodes such that it can schedule a message passing over the links. For instance, the scheduler needs to know how much resource is available in each link for scheduling a particular message. In case of the multi-hop FTT-SE architecture, considering one master node to perform the scheduling process may not be efficient, as shown by Yekeh *et al.* [18]. Thus, it may be worth to have several master nodes to organize the message transmission. As a result of having multiple master nodes, the challenge is to make consistent schedulers in the master modules in order to build a collaboration among them for scheduling and forwarding the traffic.

In order to tackle the mentioned challenges we propose two solutions for the multi-hop FTT-SE protocol. In the first solution, we propose to connect several switches in a tree topology, and to connect a master node to each switch in order to control the traffic transmission in that switch. We call this architecture the *multi-master* architecture. For the sake of timeliness guarantee, a schedulability analysis for the traffic transmitted through the network is demanded. Besides, it is important to satisfy the timing properties of the traffic when reserving the resources for them. This means that, it is important to reserve adequate resources to have a schedulable system. Therefore, we provide a response time analysis for the traffic transmitted through the switches in this architecture. We validate the analysis using network examples and we evaluate the architecture with a simulation technique. In order to improve the solution with respect to the resource usage, we propose another solution to forward the traffic in the multi-hop FTT-SE architecture. We again consider an architecture of multiple switches connected in a tree topology. However, instead of using one master node per switch, we propose to assign one master node per group of switches. This group of switches is identified by a number of switches that share a parent switch and it is called a *cluster*. Thus, each master node is responsible to coordinate the traffic of its cluster. We call this architecture *hybrid*

as it is a combination of the multi-master and single-master architectures. The single-master architecture was already presented by Marau *et al.* [19], hence it is not the contribution of this dissertation. We present a response time analysis for this architecture and compare it with the multi-master architecture in terms of resource utilization. We show that the hybrid FTT-SE architecture performs better with respect to resource utilization compared to the multi-master and the single-master architectures. We use the response time analyses for this purpose. Moreover, as the hybrid architecture is more promising, we focus on it and we propose a method to define the Maximum Transmission Unit (MTU) for each individual message such that the response time of the messages becomes shorter. The MTU is the maximum data size that an Ethernet frame can carry and it can have a significant impact on the delay of the message that is fragmented into several smaller packets. This contribution answers part of research question RQ1. In short, we present the following sub-contributions:

1. Proposing the multi-master and the hybrid architectures.
2. Presenting a schedulability analysis of the traffic.
3. Comparing the proposed architectures with respect to their resource utilization.

   The above sub-contributions are presented in the following publication:

   Mohammad Ashjaei, Moris Behnam, Luis Almeida, Thomas Nolte, *"Performance analysis of master-slave multi-hop switched Ethernet networks"*, In the 8th IEEE International Symposium on Industrial Embedded Systems (SIES), 2013, June.
4. Improving the response time of messages by configuring the MTU of each individual message.

1.3.2 Multi-hop communication in the HaRTES architecture

Although the FTT-SE and HaRTES architectures are similar in many aspects, they have some fundamental dissimilarities in the implementation of the master module. The master functions are implemented inside the HaRTES switch instead of within a particular node connected to the Ethernet switch. Therefore, providing different protocols for the HaRTES architecture might be more suitable. In particular, having multiple master modules in the multi-hop HaRTES architecture is inevitable as they are residing inside the HaRTES switches.

Considering multiple masters in the architecture arises similar challenges as in the multi-hop FTT-SE network. However, the HaRTES switch has more flexibility compared to COTS switches. For instance, the HaRTES switch contains a memory pool to buffer incoming messages. Moreover, it integrates a dispatcher per output port to control the transmission in different time windows, thus it can perform traffic shaping on messages sent asynchronously by nodes. This is not possible in the FTT-SE architecture since this feature is absent or disconnected from the FTT master control in COTS switches. Considering these features, we propose two scheduling methods in the context of the multi-hop HaRTES architecture. In the first solution, the traffic is scheduled and forwarded hop by hop, which is called *Distributed Global Scheduling* (DGS) as the scheduling method is distributed over several switches in the route of the traffic. In the DGS method each switch is responsible to schedule incoming messages to be forwarded to the next switch in the route of the messages within time-slots. This method is scalable, but may make a large response time for messages due to buffering them in each switch. Therefore, in the second solution we propose a change in the internal queues of the HaRTES switch, which instead of using FIFO they are altered to priority-based. We call this method *Reduced Buffering Scheme* (RBS) in which the messages are not buffered in every switch in the route of a message, instead they are forwarded as long as the resource reservation for the message allows such a forwarding within the current time-slot. We show that the RBS method is suitable for high priority messages as they can be delivered with shorter response time due to the priority-based queuing. In addition, we propose response time analyses for both RBS and DGS and we compare them using this analysis. We show that RBS delivers messages in a shorter amount of time in most of the cases compared to DGS. As RBS is more promising compared to DGS, we improve the response time of messages when using RBS by configuring the MTU for each individual message. This contribution answers part of research question RQ1. To summarize, we present the following sub-contributions:

1. Proposing the DGS and RBS scheduling methods.

2. Presenting schedulability analyses for the traffic transmitted through the multi-hop HaRTES network using RBS and DGS.

   The above sub-contributions are presented in the following publications:

Mohammad Ashjaei, Paulo Pedreiras, Moris Behnam, Reinder J. Bril, Luis Almeida, Thomas Nolte, *"Response time analysis of multi-hop HaRTES Ethernet switch networks"*, In the 10th IEEE International Workshop on Factory Communication Systems (WFCS), May, 2014.

Mohammad Ashjaei, Moris Behnam, Paulo Pedreiras, Reinder J. Bril, Luis Almeida, Thomas Nolte, *"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), August, 2014.

3. Comparing the scheduling methods in terms of response time of messages using the response time analysis as well as simulation.

   This contribution is presented in the following publication:

   Mohammad Ashjaei, Luis Silva, Moris Behnam, Paulo Pedreiras, Reinder J. Bril, Luis Almeida, Thomas Nolte, *"Improved message forwarding for multi-hop HaRTES real-time Ethernet networks"*, In the Journal of Signal Processing Systems (JSPS), May, 2015.

4. Improving the response time of messages scheduled by RBS by configuring the MTU of individual messages.

   This contribution is presented in the following publication:

   Mohammad Ashjaei, Moris Behnam, Luis Almeida, Thomas Nolte, *"MTU configuration for real-time switched Ethernet networks"*, In the Journal of Systems Architecture (JSA), April, 2016.

1.3.3 Dynamic reconfiguration

Both FTT-SE and HaRTES architectures provide online scheduling of traffic, hence they can adapt to run-time changes based on the application needs or changing application environment. An admission control is foreseen in the master node in order to make sure that online change requests comply with minimum application requirements. In case of the HaRTES switch the admission control is implemented inside the switch. The main duty of the admission control is to verify feasibility of a requested change, while it maintains the continued timely behavior during and after reconfiguration. Although both networks technologically handle dynamic reconfiguration, protocols to actually perform dynamic reconfiguration in the multi-hop architectures are lacking. The main issue is to find out where and how the admission control should

check whether the change is feasible. In particular, in the multi-hop architecture there are several master modules, thus several admission controls exist. Therefore, the decision making for the feasibility check can be distributed over several admission controls. In case of a distributed admission control the main challenge is to synchronize their decisions as well as applying the change consistently. In this contribution, we propose a dynamic reconfiguration protocol for each network, i.e., one for the FTT-SE network and another for the HaRTES network. We consider the traffic scheduling and forwarding algorithms such that the reconfiguration protocols are compatible with them. We prove the correctness of the reconfiguration protocols using a model checking technique and we perform a set of experiments to show their performance. This contribution answers research question RQ2. In short, the sub-contributions are stated as follows:

1. Proposing dynamic reconfiguration protocols for the multi-hop FTT-SE and HaRTES architectures.
2. Prove correctness and conduct a performance study of the dynamic reconfiguration protocols.

The above sub-contributions are presented in the following publications:

Mohammad Ashjaei, Yong Du, Luis Almeida, Moris Behnam, Thomas Nolte, *"Dynamic reconfiguration in HaRTES switched Ethernet networks"*, In the 12th IEEE World Conference on Factory Communication Systems (WFCS), May, 2016.

Mohammad Ashjaei, Paulo Pedreiras, Moris Behnam, Luis Almeida, Thomas Nolte, *"Dynamic reconfiguration in multi-hop switched Ethernet networks"*, In the 6th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES), October, 2014.

1.3.4 End-to-end resource reservation

Many industrial systems are in fact distributed real-time systems. This means that the applications in such systems span over more than one processor communicating via a network. The resource reservation for these applications include reservations on both computation as well as communication resources. Such reservation is called end-to-end resource reservation. In this scope, another trend is towards using the resources efficiently by adapting to the run-time environment. Such reservation techniques can guarantee minimum resources

for different applications, thus supporting online adaptation and reconfiguration when extra resource capacity is available while meeting minimum application requirements. In this contribution we extend our results inherited in the previous solutions to allow for multi-hop communication. We add nodes and tasks that are executing in the nodes into the picture and provide a new end-to-end resource reservation framework for distributed real-time embedded systems. We define a *transaction* comprising several real-time tasks and messages that make a data chain. In this model several tasks in one transaction are allowed to be executed in one node. Moreover, tasks and messages can have different activation patterns, including periodic or sporadic activations, which are commonly used in industrial control systems. We also define an *application* that consists of a set of transactions. The common aspect among the transactions within one application is that they belong to one functionality in the distributed system, hence they share resources (both in the computational nodes and the communication). For instance, a cruise control functionality in a modern car is considered as an application in this framework in which there are several transactions (e.g., control loops). In addition to the framework, we present an end-to-end timing analysis to determine the schedulability of the transactions given their reserved resources. The analysis computes not only the end-to-end response times, but also end-to-end data path delays. In particular, it computes the *data age* and the *reaction* delays [20], which are of importance in, e.g., control systems and body electronics. We provide a proof of concept for the framework using an automotive case study. This contribution answers research question RQ3. To sum up, the sub-contributions are stated as follows:

1. Presenting a new end-to-end reservation framework tailored for distributed embedded systems.

2. Presenting timing analysis for transactions, including end-to-end response time, age and reaction delays.

The above sub-contributions are presented in the following publication:

Mohammad Ashjaei, Saad Mubeen, Moris Behnam, Luis Almeida, Thomas Nolte, *"End-to-end resource reservations in distributed embedded systems"*, In the 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), August, 2016.

1.3.5 Personal contributions

The research presented in this work is done in collaboration with researchers at University of Porto, University of Aveiro and University of Eindhoven. I am the main contributer and the first author of all included papers. Prof. Thomas Nolte, Dr. Moris Behnam and Prof. Luis Almeida are my supervisors. They all contributed in discussions and reviewing the solutions. Moreover, Prof. Paulo Pedreiras and Prof. Reinder J. Bril contributed in reviewing and discussions on the solutions and analyses. Luis Silva contributed in implementing one of the solutions (the RBS traffic forwarding over HaRTES architecture) on the HaRTES switch and performing the experiments on the hardware. Dr. Saad Mubeen contributed in reviewing and discussions on the end-to-end timing analysis for distributed embedded systems. I was supervising Yong Du during his master thesis. He proposed a solution for dynamic reconfiguration in the multi-hop HaRTES architecture. I modified the solution in several directions, such as solving conflict cases when using the dynamic reconfiguration. I also presented a formal analysis for the solution to prove its correctness. In addition, I performed the related experiments and evaluations for the dynamic reconfiguration protocol.

1.4 Research methodology

In order to prove the thesis statement, we adopted and followed a deductive research methodology. In the deductive research method a hypothesis based on an existing theory is developed. Then, a strategy to test the hypothesis is followed to reach a confirmation or a rejection [21]. This method of research, which is also called top-down approach, can be formulated to define a thesis statement and to make a strategy to prove it. The flow of the research in this work is illustrated in Figure 1.1.

First we defined the main thesis statement by reviewing the state of the art and finding limitations in the context of switched Ethernet protocols for multi-hop real-time communication. Then, we identified the research questions after analyzing the possible challenges. We explored the possible solutions to answer the research questions. In order to propose a solution we got help from discussions during several technical meetings and conferences. After having a solution, we validated it using different techniques depending on the research question. To answer RQ1, the main aim is to find solutions to extend the existing protocols. We used mathematical models in the form of response time analysis together with simulation to validate the proposed solutions. However,

to answer RQ2 we used a model checking technique to prove the correctness of the proposed reconfiguration protocols. Finally, to answer the last research question RQ3 we used mathematical models to analyze the end-to-end resource reservation model as well as an experimental study on a case study to show the practicality of the model. In case the solution was not desirable according to the validation method, we tuned or completely altered it. After achieving a desirable solution we finalized the solution by publishing scientific reports or papers. As the proposed solutions were not optimum considering resource usage, cost and complexity of their implementations, we studied their improvement points. Focusing on the improvement points we proposed new solutions for the identified problems. We performed the same methodology to answer all research questions.

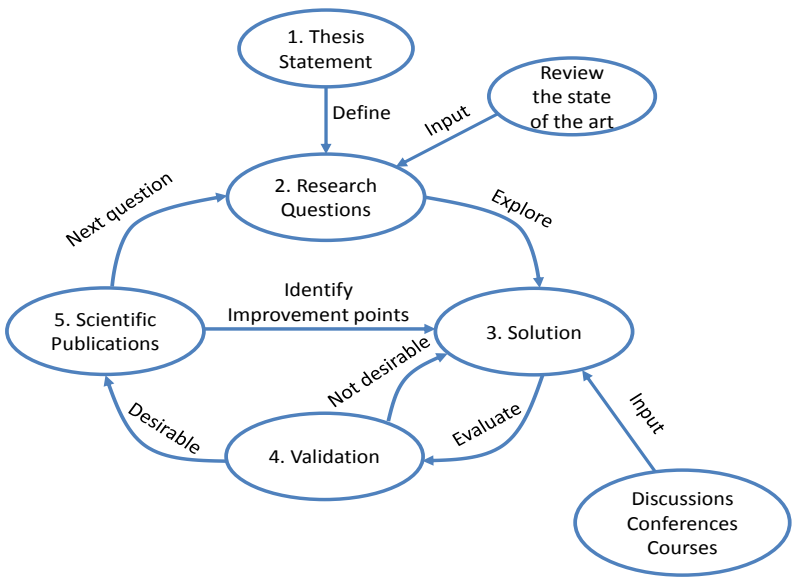

Figure 1.1: The flow of research process.

1.5 Thesis outline

The rest of the dissertation is organized as follows.

- **Chapter 2 - Background and Prior Work:** This chapter is twofold containing required background and prior work relevant to this dissertation. The background part describes the required information related to real-time embedded systems and communication protocols. In addition, the prior work discusses the existing research literature that is more related to the contributions.

- **Chapter 3 - System Model:** This chapter provides the system model, including the network and traffic model, that is the basis for the rest of the work. It also presents a complete description of the single-switch FTT-SE and HaRTES architectures as part of the network model.

- **Chapter 4 - Multi-Hop FTT-SE Architecture:** This chapter presents two solutions for extending the single-switch FTT-SE protocol to provide multi-hop communication. Moreover, it presents a schedulability analysis of the traffic according to a response time analysis for each solution. The chapter also presents a comparison study between the solutions on the resource utilization to propose a suitable solution. The contents of this chapter appeared in a publication [22].

- **Chapter 5 - Multi-Hop HaRTES Architecture:** This chapter proposes two solutions to extend the single-switch HaRTES architecture. Moreover, it presents a schedulability analysis of the traffic based on a response time analysis for each solution. The analysis helps to compare the solutions in order to propose a method with shorter response time for traffic, hence to utilize less resources. The contents of this chapter appeared in several publications [23, 24, 25].

- **Chapter 6 - MTU Configuration in FTT-SE and HaRTES:** This chapter proposes methods to configure the MTU of individual messages in both FTT-SE and HaRTES networks such that the response time of messages become as short as possible. The contents of this chapter appeared in a publication [26].

- **Chapter 7 - Dynamic Reconfiguration in FTT-SE and HaRTES:** This chapter presents dynamic reconfiguration protocols for the multi-hop FTT-SE and HaRTES networks. It also provides a validation for the

proposed protocols according to a model checking technique. A set of experiments is presented to show the performance of the protocols. The contents of this chapter appeared in several publications [27, 28].

- **Chapter 8 - Distributed Resource Reservation:** This chapter takes a holistic view on the distributed embedded system. It presents a new framework for end-to-end resource reservation in distributed embedded systems. Moreover, the chapter presents a timing analysis consisting of end-to-end response time analysis, data age delay and data reaction delay computation. The practicality of the framework and analyses are validated using an automotive case study. The contents of this chapter appeared in a publication [29].

- **Chapter 9 - Conclusions and Future Work:** This chapter concludes the work by summarizing the presented contributions and discussing the future directions.

Chapter 2

Background and Prior Work

This chapter introduces the background concepts that are required for understanding the contributions of the dissertation. In addition, the work builds upon a large body of prior work in the area of real-time communications. Therefore, this chapter also reviews the relevant state of the art regarding the contributions that are presented in the rest of the book.

2.1 Real-time systems

An *embedded system* is a microprocessor-based system that is designed to perform a specific function [30]. They are mostly components of a larger system, where they often interact with the environment through sensors and actuators. The embedded systems are found in almost all electronic devices ranging from simple items, such as home appliances and alarm clocks, to more sophisticated objects, like modern cars and industrial control systems. Besides the high number of embedded systems that has sharply increased in the last decade, the software that runs on the embedded systems has increased drastically in both size and complexity. For instance, around 100 million lines of code are running in several embedded systems to get a premium car out on the driveway [31].

For some of the embedded systems, not only the logical correctness of the functionality is important, they also have to produce the logical results within a specified time, which is known as a *deadline*. This type of embedded systems is called *real-time* embedded systems. Depending upon the consequences that may occur because of a missed deadline, a real-time system can be categorized

into two groups: Hard and Soft. If the results of an embedded system are produced after its specified deadline and that causes system failure, the system is called a hard real-time system. Whereas, if the consequences of missing a deadline cause a performance degradation but still it has some utility for the system, the system is called a soft real-time system. For example, the airbag control system in a car is a hard real-time system, while video and audio streaming typically is denoted as soft real-time systems.

2.1.1 Real-time tasks

In real-time systems, a task is defined as a computational entity, which performs a certain activity. A real-time task produces results within a deadline that is defined for the task. Real-time systems are commonly limited in resources, such as memory, computation and communication, that in turn impose strict requirements on the tasks. These tasks can be triggered (activated) by events, which can be events generated by time, other tasks or external events. The instant in which a task is triggered is called the *task arrival*. Moreover, the duration of time between two arrivals of the same task is called the *task inter-arrival time*. Based on the regularity of the task arrivals, tasks in the real-time system can have different activation patterns, that can be periodic, sporadic or aperiodic. Periodic tasks are activated with a fixed task inter-arrival time (known as *period* of the tasks), while sporadic tasks are not activated periodically, but minimum inter-arrival times are known. Finally, aperiodic tasks have unpredicted arrival times, normally without specified deadlines. Figure 2.1 shows the difference between the mentioned activation patterns for a task.

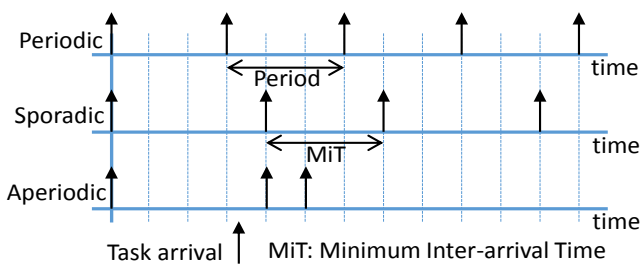

Figure 2.1: Different activation patterns for a task.

The choice of developing different tasks depends upon the characteristics and nature of the environment in which the tasks are to operate within. Tasks that are taking care of measurements or control processes are normally periodic tasks, whereas for handling events with a known minimum inter-arrival time due to physical limitations, such as pressing a button, sporadic tasks are implemented. Aperiodic tasks are implemented for handling events where it is not possible to guarantee minimum inter-arrival times. Such tasks in a real-time system should not interfere with other tasks since their interference would prevent guaranteeing the performance of the system.

2.1.2 Real-time scheduling

Several tasks can be executed with a specified order in a real-time system. This order is dictated by a *scheduler* according to an algorithm. The main aim of the scheduler is to execute the tasks in a way that all tasks meet their deadlines. Several scheduling algorithms are proposed with different characteristics [32]. A taxonomy of real-time task scheduling is illustrated in Figure 2.2.

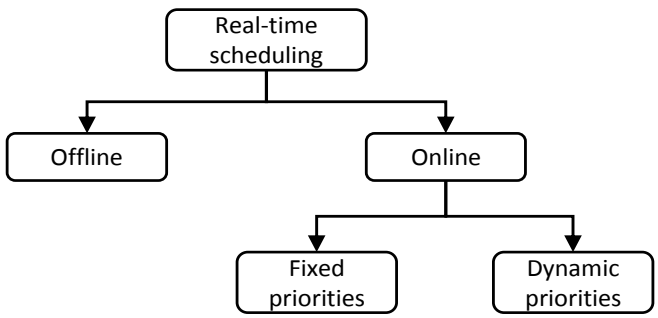

Figure 2.2: A taxonomy of real-time task scheduling.

The scheduling of real-time tasks can be done offline or online. In the former, the order of the tasks for execution follows a predefined schedule at each point in time. However, in online scheduling, the scheduler decides about execution of tasks, at each point in time, according to the *priority* of the tasks. Therefore, a priority is assigned to each and every task in the system, which can be either fixed or dynamic during run-time. The priority of a task can

be dynamic in the sense that each time a task arrives for execution it can exhibit a different priority. Two common scheduling algorithms [1] in uniprocessor real-time systems are Rate Monotonic (RM) for fixed-priority scheduling and Earliest Deadline First (EDF) for dynamic priority scheduling [35]. According to RM, priorities for tasks are assigned monotonically with respect to the period of tasks, where the shorter period gets a higher priority. Then, these priorities are fixed during run-time of the system. EDF assigns the higher priority to tasks with the earliest deadline relative to current time, thus the priorities are dynamically changing depending on the current set of tasks that are ready for execution. In addition, in real-time systems, tasks can be interrupted in the middle of their execution, allowing a higher priority task to access the processor as soon as possible. Such a system is called a *preemptive system*. If the system does not allow interrupting the execution of tasks, it is called a *non-preemptive system*. Figure 2.3 illustrates preemptive and non-preemptive execution of tasks.

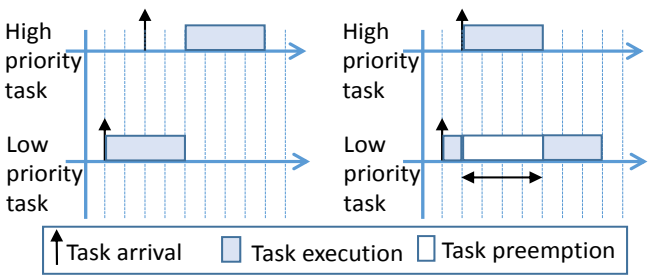

Figure 2.3: Preemptive and non-preemptive execution.

2.1.3 Share-driven scheduling

An alternative way for scheduling tasks is to allocate a fraction of the processor to them [36]. As mentioned before, aperiodic tasks have unknown behavior which should be isolated from other real-time tasks to guarantee the performance of the system. Using share-driven scheduling it is possible to prevent interference between aperiodic tasks and other tasks beyond an admissible

[1] In the literature [33, 34], these algorithms serve the purpose of both priority assignment for tasks as well as scheduling of tasks.

level. The service to provide share-driven scheduling can be achieved by the usage of *servers*. A server can be seen as a periodic or sporadic task whose duty is to serve requests of execution of aperiodic tasks. Similarly to regular tasks, servers can be scheduled according to the fixed-priority scheduling policy or the EDF scheduling algorithm. A server is characterized by its capacity, replenishing policy and priority. There are many types of servers introduced in the literature with different improvements with respect to their performance. Common servers for fixed-priority scheduling policies are *Polling Servers* [37], *Deferrable Servers* [38] and *Sporadic Servers* [37], whereas common servers in dynamic priority systems are *Total Bandwidth Servers* [39, 40] and *Constant Bandwidth Servers* [41]. In this section, as a simple example, the polling server is explained in more detail.

A polling server is activated periodically to serve pending requests. Figure 2.4 shows a polling server that services an aperiodic task. The capacity of the server shows how much time of the processor is allocated to the server. At the beginning of the server activation, the capacity is replenished to its defined value. If there are any pending requests, the server handles them as long as its capacity allows, otherwise the server suspends itself until the next period. It should be noted that if there is no pending request, the server drops its capacity to zero. Therefore, if a request for a task arrives just after the server suspension, it has to wait for the next period, where the server replenishes the capacity to its full value. Moreover, the server has a priority, that allows other higher priority tasks to preempt the server if needed. In Figure 2.4, the task is activated at time unit 3, where the server capacity is already exhausted. Thus, it has to wait for the next period of the server in which the task can be executed and uses the server capacity. Note that in this example the task execution time is 4 time units and it is equal to the server capacity.

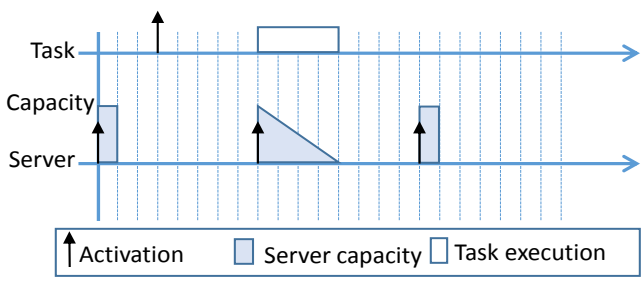

Figure 2.4: A polling server example.

A deferrable server improves the average response time of aperiodic tasks compared to a polling server. The main difference is that a deferrable server preserves its capacity if there is no pending request.

2.1.4 Hierarchical scheduling

A real-time system can be decomposed to several sub-systems each of which can be using a scheduling policy that is more suitable to the system. Therefore, server-based scheduling can be adopted for each sub-system to schedule it separately from the other parts of the system. This way a hierarchy of schedulers can be constructed, as shown in Figure 2.5 as an example of hierarchical scheduling utilizing two levels of schedulers.

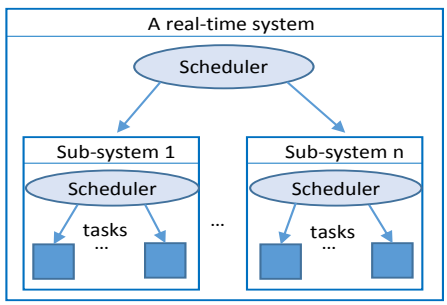

Figure 2.5: A two-level hierarchical scheduling.

For each sub-system a server is developed to serve the tasks within the sub-system. The scheduler can be based on any desirable policy that is suitable for the set of tasks. Moreover, the schedulers can be different in each sub-system as they are executed separately. A global (system-level) scheduler schedules the servers, again with a suitable policy. The hierarchy can be more than two levels if it is required. Servers in the hierarchical schedulers can be deferrable [42] or polling [43]. Moreover, hierarchical scheduling frameworks based on EDF are also presented in the literature [44, 45, 46].

2.2 Real-time communication

Many embedded applications are implemented using a distributed architecture where the functionality is distributed over many processors (nodes) interconnected by a communication system (network). Such systems are commonly known as distributed embedded systems, where the processing nodes require exchanging information. In case of real-time distributed systems, the temporal behavior depends on several items including timing requirements in both processing software in the nodes and information exchanged over the network. In such a distributed system, the communication system has to deliver information within specific timing constraints, which is known as a *real-time communication* system.

A good example is a modern car where many software functions, such as cruise control and anti-lock braking, are distributed among some of the 70 to 100 Electronic Control Units (ECUs) that are connected via different networks [47]. Looking specifically at the cruise control mechanism, there are several sensors needed to measure speed, throttle value and brake pedal position in the vehicle. The information from these sensors is sent to a central computing unit via a communication system. A control loop in the unit provides a control value to be submitted to the actuators such as the throttle and the brake. The success of the system depends on the time it takes from the sensor measurement to the actuation.

The information within a distributed system is transmitted by means of *messages*. Similar to real-time tasks, real-time messages should be delivered within their deadlines. Moreover, messages can be activated periodically, sporadically or aperiodically, following the same taxonomy as in real-time tasks, which will be explained in the next subsection.

2.2.1 Event-triggered and time-triggered communication

From the the point of view of how the communication is triggered, communication services can be classified in two different paradigms: *event-triggered* and *time-triggered* communication [48]. In time-triggered communication, all nodes have a common time reference established using a time synchronization method. The applications send messages at certain predefined points in time in a periodic manner. This paradigm can provide a great determinism as the schedule of message transmission can be built in advance, thus it prevents simultaneous transmission which may lead to collision. Based on the event-triggered communication paradigm, messages are sent by the node upon

the occurrence of an event, such as termination of a task or an interrupt signal. These events may occur at any time instants, however to define a worst-case scenario a minimum inter-arrival time can be defined between consecutive events, i.e., sporadic messages.

Each paradigm has some advantages making them suitable for specific applications. In general, the time-triggered approaches are suitable for control systems as they deal with periodic sampling and actuation, hence periodic messages. In contrast, the event-triggered approaches are more adequate to handle sporadic messages like a sensor changing value in automation. In some applications both periodic and sporadic activities are occurring, thus a combination of the event-triggered and the time-triggered paradigm is desirable in order to gain from their respective profits. A usual technique applied in this situation is to use communication slots containing two consecutive phases (or windows) dedicated to each type of transmission. The size of the phases can be selected according to the need of a particular application. This way, each type of traffic is forced to respect the associated phase for transmission, hence a temporal isolation among them is guaranteed. This method has been used in many communication protocols such as FlexRay [49], FTT-CAN [50] and FTT-Ethernet [51].

2.2.2 Message scheduling

Commonly, distributed systems are built over a shared medium network where the nodes exchange data messages. In order to maintain the timing requirements, access to the communication network should be properly scheduled. Therefore, scheduling of messages is required. The scheduling of messages can be done using the scheduling policies introduced in the microprocessor domain, however depending on the system requirements along with the hardware and software support. These scheduling policies include offline scheduling and online scheduling with fixed or dynamic priority assignments [48]. The scheduling decision in offline scheduling is at compile time using a dispatching table, where the time of transmissions for each message is stated. Therefore, a complete knowledge about the characteristics of messages is required during the design phase, including the deadlines, precedence constraints and particularly arrival times. This scheduling method requires that all nodes have the same view of time, hence a synchronization among the nodes is mandatory. An example of a network protocol that uses offline scheduling is Time-Triggered Ethernet [4], where messages are allocated to time-slots. On the other hand, the scheduling decision in online scheduling is performed during run-time by

selecting one message out of several ready messages at each point in time. For this purpose, a priority is assigned to each message, which can be fixed or dynamic during run-time, helping the scheduler to pick the higher priority message out of several. An example of using fixed-priority scheduling of messages in networks is the Controller Area Network (CAN) [52, 53].

Despite similarities of the network domain and the microprocessor domain, such as using scheduling policies and message activation patterns (periodic, sporadic or aperiodic), there are additional challenges in real-time distributed systems regarding the scheduling process. These challenges are imposed by the nature of the distributed architecture as follows. One key element in scheduling is to keep track of available resources, which is not always easy in distributed systems. In fact, the knowledge of the system state, in particular network and nodes, is not always available. This makes the scheduling decision to be made based on incomplete information [54]. Moreover, in the context of scheduling in microprocessors, preemptive scheduling is commonly used as, in average, it provides a higher schedulability ratio compared to non-preemptive scheduling [55]. However, in communication systems a packet normally cannot be preempted during its transmission. Packets are the result of a message fragmentation, hence preemption can be applied at message level.

2.3 Schedulability analysis

As the major requirement of real-time systems is to provide actions in a timely manner, an accurate prediction of timing behavior is required. In order to guarantee that the actions meet their respective deadlines, schedulability analysis techniques have been developed [34, 56, 33]. This section describes some of the analysis techniques commonly used in real-time systems.

2.3.1 Utilization-based tests

Processor *utilization factor* U is a fraction of time that a processor spends on execution of a set of real-time tasks [57]. The utilization factor for N tasks is computed as below, where C_i is the execution time and T_i is the period of task τ_i.

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \tag{2.1}$$

If the utilization factor of a set of tasks becomes greater than one, the set cannot be schedulable by any scheduling algorithm. Liu and Layland [57] (and even earlier by Fineberg and Serlin [58] without mathematical formulation) a utilization-based test was presented for periodic tasks that are scheduled by RM. The test is under assumption of fully preemptive tasks and the task activations are at the beginning of their period. The test is presented in Equation 2.2 that shows a set of N tasks is schedulable if the following inequality holds.

$$U \leq N \times (2^{\frac{1}{N}} - 1) \tag{2.2}$$

Moreover, Liu and Layland [57] showed that RM is an optimal scheduling algorithm among all fixed-priority scheduling algorithms, which means no other fixed-priority algorithm can schedule a task set if RM cannot. For a task set that is scheduled by EDF, the test is presented by Liu and Layland [57], where the utilization should be less or equal than one.

2.3.2 Response time analysis

One of the schedulability analysis techniques is based on calculating the worst-case response time of tasks/messages and comparing that to their deadlines. the response time of a message is the time interval between the activation time of a message in the source node and the reception time in the destination node. This technique is known as Response Time Analysis (RTA) [59]. The RTA technique is a well-known method in the real-time community and it has been used in many real-time domains including microprocessors, multi-core systems and distributed systems. When using this technique the response time of a task/message is calculated iteratively with a pseudo-polynomial run-time complexity, with the aim to find the worst-case scenario for the task/message. Also, interference of other tasks/messages and their activation frequencies are taken into account in the computation.

In the context of single-core microprocessors and assuming a fixed-priority preemptive scheduling, the worst-case scenario for a task occurs when all higher priority tasks are released simultaneously with the task, known as the *critical instant* for the task [35]. In the above mentioned situation (i.e., at critical instant), the response time R_i of a task τ_i is calculated by the sum of the task computation time C_i, the interference from higher priority tasks denoted by I_i and blocking due to sharing a common resource with lower priority tasks, that is denoted by B_i. Equation 2.3 shows the response time calculation for task τ_i.

$$R_i = C_i + B_i + I_i \tag{2.3}$$

Moreover, the interference from higher priority tasks is calculated in Equation 2.4 considering all activations of higher priority tasks during the response time of task τ_i, where $hp(i)$ is the set of tasks with priority higher than that of τ_i and T_j is the period of those tasks.

$$I_i = \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \tag{2.4}$$

Combining equations 2.3 and 2.4 derives an iterative equation shown in Equation 2.5, where $R_i^0 = C_i$ and the result is derived when $R_i^{n+1} = R_i^n$.

$$R_i^{n+1} = C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{R_i^n}{T_j} \right\rceil C_j \tag{2.5}$$

Note that the response time calculation presented above is valid for fixed-priority preemptive scheduling. In case of EDF scheduling, the computation is different [60].

The same algorithms can be applied for messages considering different way of dealing with interference, blocking and critical instants. Davis *et al.* [61] presented the calculation of worst-case response time for messages in CAN networks. In the area of Ethernet AVB networks, Bordoloi *et al.* [62] computed the response time of traffic class A and B. Furthermore, Marau *et al.* [19] and Santos *et al.* [63] used this technique to compute the worst-case response times of Ethernet messages in a switched Ethernet network.

2.3.3 Network Calculus

Network Calculus [64] is a framework for analyzing deterministic queuing systems based on *min-plus* algebra. The traffic transmitted to a network is affected by constraints imposed by the network components such as link bandwidth and traffic shapers. The constraints associated with the traffic flow are expressed by an *arrival curve*, whereas the availability of resources such as crossed nodes is described by a *service curve*. The delay bound which represents an upper bound to the worst-case response time and the backlog that represents an upper bound to the maximum queue length are computed using the arrival and service curves. Figure 2.6 shows the service and arrival curves.

In the simplified case, the arrival curve can be upper bounded by $\alpha(t) = b + r \times t$ such that b is the maximum burst and r is the rate of traffic. Moreover, the service curve for a node is $\beta(t) = max(0, R \times (t - \Delta))$, where R is the rate of service and Δ is the latency imposed by the node. Then, the delay bound is

the maximum distance between the two curves as presented in Figure 2.6. Note that the illustration is a simplified version of the delay calculation, however the arrival and service curves can be more complex, e.g., step-like curves.

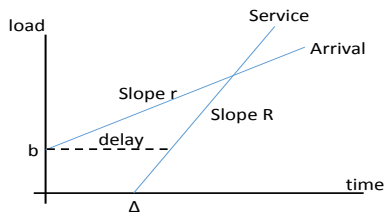

Figure 2.6: The service and arrival curves in network calculus.

In the context of Ethernet networks, several approaches use Network Calculus to compute the delay of traffic. Mifdaoui *et al.* [65] used Network Calculus to analyze the delays of the traffic in a single-master and multi-switch network topology using the FTT-SE protocol. Moreover, Network Calculus is used in networks of standard Ethernet switches presented by Zhang *et al.* [66]. Charara *et al.* [67] presented three methods to derive the traffic delays in a multi-hop AFDX network. These three methods include Network Calculus, network simulation and model checking, among which Network Calculus exhibits a higher pessimism. Focusing on Ethernet AVB, Queck [68] utilized Network Calculus to derive traffic delays. Moreover, Manderscheid *et al.* [69] presented a worst-case delay verification of in-vehicle Ethernet networks using the same analytical framework to generate upper bounds and checking them against experiments in worst-case scenarios. A different approach is followed by Lenzini *et al.* [70] and Schmitt *et al.* [71] that derives end-to-end delay bounds for a single flow in FIFO multiplexed sink-tree networks using a modified Network Calculus framework. Furthermore, Lenzini *et al.* [72] utilized the mentioned method to investigate an admission control in sink-tree networks.

2.3.4 Trajectory approach

The trajectory approach [73] has been developed to compute the upper bound on delay of the traffic. It is based on analysis of the worst-case scenario experienced by a message on its trajectory from the source node to the destination

node. According to this technique, the delay experienced by a message from its release in its source node until its reception in its destination node is the sum of transmission delay and the time the message is spent in each node. Note that a message can traverse through several intermediate nodes (these nodes can be translated to switches in the context of switched networks). The transmission delay on links is upper bounded by L_{max} that depends on the transmission speed, packet length and physical limitations.

The delay of crossing a node depends on the status of the output queue of the node. Assuming that the queue is FIFO, the arrival message should wait for all messages ahead of that in the queue. In order to compute the waiting time in the crossing nodes, the trajectory approach introduces a *busy period* level $\mathcal{L}$. A busy period [74] is an interval of time between two consecutive *idle times*. An idle time is a time when all previously arrived messages have been processed.

The trajectory approach calculates the latest starting time of message m in its last node considering that the message is generated at time t in its source node. Then, it derives the delay. The trajectory approach has been used in AFDX networks as a tighter analysis compared to the Network Calculus framework [75, 76, 77]. However, in the work presented by Kemayo *et al.* [78] some corner cases are found in which the improved trajectory approach introduces some optimism, even though these corner cases do not exist in any AFDX configuration. The solution to resolve this problem is proposed by Li *et al.* [79]. Moreover, the trajectory approach is used to compute the worst-case backlog in the AFDX network [80].

2.3.5 End-to-end delays

In a real-time system tasks may be executed with precedence constraints. This means that a task is only allowed to be executed after the finishing time of its previous task (among which the precedence constraint exist). Moreover, a task may get input data from the execution of another task even when they are executed independently. This makes a *task chain*, where each task may be triggered by its previous task or independently. When a system is modeled with task chains then the schedulability of the system can be determined by calculating end-to-end response time and end-to-end delays in each chain and comparing them with the corresponding deadlines. In order to understand the difference between the end-to-end response time and end-to-end delays consider two task chains as shown in Fig. 2.7. Both task chains contain three tasks and each task is assumed to have the Worst Case Execution Time (WCET) of 1 ms. The tasks have equal priorities and communicate with each other by

writing to and reading from the registers[2]. There is only one activating source for the first chain, shown in Fig. 2.7(a), that activates τ_1 periodically with a period of 8 ms. The rest of the tasks in the chain are activated by their predecessors. Such a chain is called a *trigger chain* [81]. The input of the chain corresponds to the data read by τ_1 from register Reg-1. The input may correspond to the data that arrives from a sensor. On the other hand, the data written by τ_3 to Reg-4 is considered as the output of the task chain. The output may correspond to the control data for an actuator. The end-to-end response time of the task chain is defined as the amount of time elapsed between the arrival of an event at the first task and production of the response by the last task in the chain. The arrival of an event corresponds to the periodic activation of task τ_1. Assuming no interferences, the end-to-end response time of the chain can be calculated by summing up the WCETs of all tasks in the chain, which is 3 ms in this example.

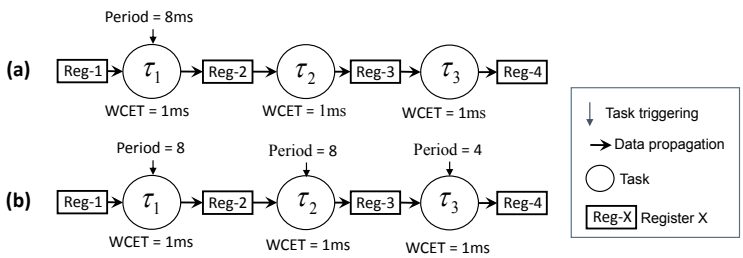

Figure 2.7: Activation patterns: (a) a trigger chain, (b) a data chain.

Now consider the task chain shown in Fig. 2.7(b). In this chain, each and every task is activated independently, that means no task is activated by its predecessor. Such a chain is called a *data chain* [81]. It should be noted that a mixed chain includes both trigger and data sub-chains. The periods of activation for tasks τ_1, τ_2 and τ_3 are 8 ms, 8 ms and 4 ms respectively. Since each task in the chain is activated independently with a different clock, there can be several time paths through which the data can traverse from the input to the output. Hence, there can be multiple outputs that correspond to one single input of the chain as shown by the uni-directional curved arrows in Fig. 2.8. As a result there are different delays between the arrival of the input data and production of the output data by the chain.

[2]A register may correspond to a port of a software component which is realized by the task at run-time.

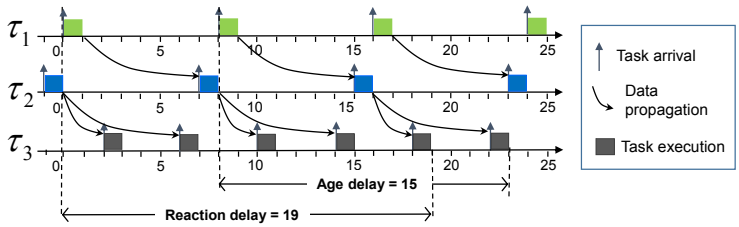

Figure 2.8: A possible execution trace for the transaction in Fig. 2.7(b).

In this work, two such delays are described, namely *age* and *reaction* [20]. The data age delay is equal to the time elapsed between the arrival of data at the input and the latest availability of the corresponding data at the output. For example, assume that the data is available in Reg-1 (input of the chain) at time 8 when it is read by the first task (τ_1). Task τ_1 writes the corresponding data to Reg-2 at time 9. The data is read by task τ_2 from Reg-2 at time 15. Task τ_2 then writes the corresponding data to Reg-3 at time 16. Task τ_3 reads the same data from Reg-3 at two different times, i.e., at 18 and 22. Moreover, task τ_3 writes the corresponding data to Reg-4 (output of the chain) at times 19 and 23. It can be seen that there are two samples of output data corresponding to one single sample of the input data. In the age delay, the longest time difference between the input data and corresponding output data is of interest. On the other hand, the data reaction delay corresponds to the earliest availability of the data at the output corresponding to the data that *just missed* the read access at the input. Assume that just after task τ_1 started its execution at time 0, new data arrives in Reg-1. Hence, the new data just misses read access of τ_1. The new data has to wait in Reg-1 until it is read by τ_1 at time 8. The earliest effect of this data appears at the output at time 19 which corresponds to the data reaction delay. In this example, the age delay is 15 ms, while the reaction delay is 19 ms.

The data age delay is important, in particular, for control applications where freshness of the data is of value. Whereas, the data reaction delay is important in the applications where the first reaction to the input is of value.

2.4 Ethernet technology

Ethernet was originally developed as a technology for computer networking. Nowadays, Ethernet is supported by a collection of IEEE standards defining different network layers. Besides evolution in the transmission speed, Ethernet evolved in the physical layer from a bus topology to a star and switching hubs

(commonly known as Ethernet switches) topology. Originally, Ethernet used a shared medium for communication, thus it created a single collision domain. This means that a message was transmitted in the medium and all nodes could receive that message. Moreover, an arbitration mechanism called Carrier Sense Multiple Access with Collision Detection (CSMA/CD) was used to overcome the collision problem. Essentially, in this arbitration, a node holds the message until the shared medium becomes idle. As all nodes are following the same rule, the collision becomes inevitable. In case of collision the transmissions involved in the collision were aborted and the nodes waited for a random amount of time, then they started retransmission of the messages. This procedure was continuing until a successful transmission, i.e., without collision. This problem of message retransmission, which leads to a low efficiency in using the bandwidth, was the main motivation to use Ethernet switches. A switch provides a private collision domain for each of its ports, hence it reduces the impact of the non-deterministic feature of the original Ethernet inherent in the CSMA/CD arbitration. When a single node is connected to each switch port through full-duplex links, a topology known as micro-segmentation, CSMA/CD is not used at all, being bypassed by the queuing policing in the switch ports. This is the most common topology used.

An Ethernet frame is composed by several fields, which is illustrated in Figure 2.9. The Preamble and SOF fields are used to synchronize the clock and presenting the start of the frame. The MAC address shows the source and destination identification of the frame. The Ethernet Type field represents the data payload encapsulation protocol. Moreover, the Payload field contains the data to be sent, that can be between 46 to 1500 bytes. The frame ends with the Frame Check Sequence (FCS) of 4 bytes length that is used for error detection. The standard defines an inter-frame gap between any two consecutive frames. This is an idle time on the Ethernet link, which corresponds to the time taken to transmit 12 bytes. The Ethernet frame was later standardized in IEEE 802.3, which brought a slight modification. The Type was replaced by a Length field (payload length) with the same size. Both frame types, referred as Ethernet-II and IEEE 802.3, coexist today and are distinguished by using Types above 1500. Other optional fields were introduced later in IEEE 802.1Q, which we do not use in this work.

In some applications the messages to be sent through Ethernet networks can be larger than the size of the payload in a frame. In such cases, a message is fragmented into several frames (packets).

Preamble (7 bytes)	SOF (1 byte)	Destination address (6 bytes)	Source address (6 bytes)	Ethernet type (2 bytes)	Payload (46-1500 bytes)	FCS (4 bytes)

Figure 2.9: The Ethernet frame.

2.4.1 Switched Ethernet

Most common switches are based on IEEE 802.1D that introduces FIFO queues for each output port. A message arrives to a switch and after recognizing the destination address, it is buffered in the output queues. The message stays in the queue for the time needed to transmit the messages ahead in the queue. According to the IEEE 802.1D standard, for each output port a limited number of parallel queues are considered for different priority levels. Messages that are queued in different priority levels are distributed based on their priorities starting from the higher priority queue. The basic structure of an Ethernet switch is depicted in Figure 2.10.

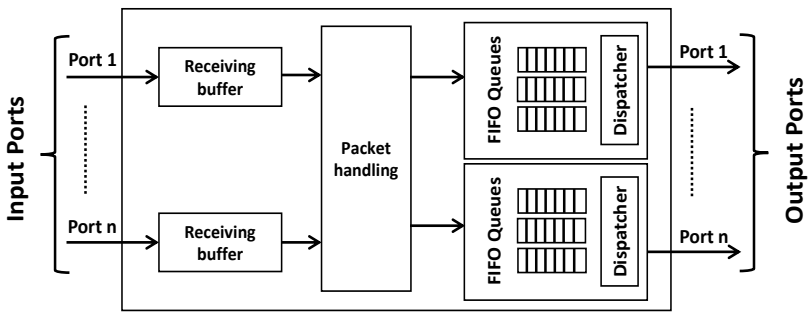

Figure 2.10: The switch internal structure.

Ethernet switches commonly support three types of message forwarding: unicast, broadcast and multicast. In unicast forwarding a message is transmitted from one source node to one destination node, directly, using the destination MAC address. Whereas, the broadcast forwarding type transmits a message from one source node to all other nodes connected to the switch simultaneously. This way the MAC address should be filled with the MAC broadcast

address so that the switch can recognize the message to be sent to all output ports. Moreover, the multicast forwarding is transmitting a message from one source node to a group of nodes connected to the switch. Only some switches support true multicasting. Those that do not, convert the multicast in a broadcast and flood the packet to all ports. The packet handling module depicted in Figure 2.10 performs message forwarding based on the MAC addresses.

Ethernet switches are categorized based on their forwarding methods. Ethernet switches can be either of store-and-forward or cut-through type. A store-and-forward switch receives a message and entirely buffers that message before forwarding it to the related output port. On the other hand, the cut-through switch starts forwarding a message as soon as it receives the header with the destination address. In the store-and-forward method, the propagation of erroneous messages can be prevented by discarding messages that were not properly received.

Using COTS switches in real-time systems may affect the possibility of guaranteeing timeliness behavior. The queues in the switch may overflow due to uncontrolled arrival of messages, which leads to messages being dropped. In addition, due to the FIFO queues, urgent messages may be blocked for a long time. The latter problem can be partially mitigated by prioritizing the messages and using separated queues for different priorities. However, the standard foresees up to eight priority levels, which is too few to support effective priority-based scheduling. Therefore, many solutions have been studied and presented to overcome these limitations. The literature on switched Ethernet is vast and there have been many works addressing its adequacy to real-time communication. In the following, some of these protocols are discussed in more detail.

2.4.2 Time-Triggered Ethernet

Time-Triggered Ethernet [4] is a protocol based on an enhanced Ethernet switch and it is optimized for time-triggered transmission. The protocol defines three different traffic classes [82], being time-triggered (TT), rate-constrained (RC) and best-effort (BE) traffic classes. The prime mode for the communication is time-triggered traffic transmission. The protocol provides short latency and jitter in transmission for TT messages. These messages are scheduled offline and the nodes are transmitting them according to a synchronized global time among the switches and nodes. If a time-slot in the communication is free of TT messages, the switch recognizes that and associates the available bandwidth to the other traffic classes. RC messages, on the other hand, have less

strict real-time requirements than TT messages. However, sufficient bandwidth must be allocated to them in order to provide defined limits for their delays. As Time-Triggered Ethernet provides deterministic communication for RC messages, timing analysis for this class of messages is required [83, 84]. The switch behaves as a classic Ethernet switch for BE messages, where no temporal guarantee is provided.

In order to obtain a seamless communication of TT messages, a clock synchronization protocol is defined. There are three different roles for switches and nodes in the network concerning synchronization: (i) Synchronization Master (SM), (ii) Synchronization Client (SC), and (iii) Compression Master (CM). The SM role is typically assigned to the nodes, while the switches get the CM role. Switches and nodes in an architecture can get the role of SC. The synchronization is performed in two steps. In the first step, SMs send a specific synchronization message to CMs. Accordingly, each CM produces a new synchronization message. In the second step, CMs send their produced synchronization messages back to the SMs and SCs. Then, a new reference time is derived using the received synchronization message in the SMs and SCs. The synchronization is done periodically during run-time of the system.

Scheduling of TT messages is a bin-packing problem, which is known to be NP-hard. The complexity of scheduling depends on different parameters in the network architecture including the constraints of the application and network topology. A scheduling algorithm based on Satisfiability Modulo Theories (SMT) solver is proposed for TT messages in Time-Triggered Ethernet networks [85]. Moreover, a scheduling approach is proposed to consider tasks into account targeting the end-to-end delays [86].

In addition, usability of Time-Triggered Ethernet in different areas is investigated. For instance, a candidate to industrial wireless networks based on the Time-Triggered Ethernet concept is proposed [87], which is complemented with security issues [88]. Moreover, practicality of using deterministic communication based on time-triggered architectures is investigated [89].

2.4.3 AFDX

Avionics Full DupleX Switched Ethernet (AFDX) [9] is developed as a network communication specification with enhanced forwarding. AFDX has been used mainly in avionics. According to the protocol, a Virtual Link (VL) is defined as a logical communication channel between one source node and one or more destination nodes. Each VL is characterized by a Bandwidth Allocation Gap (BAG), which is a minimum time interval between consecutive frames.

In order to enforce this characteristic in the source nodes a traffic shaper is developed to provide a deterministic communication behavior.

Offline planning for the traffic transmission is required to enforce the end-to-end delay guarantees which can be provided using an end-to-end delay analysis (e.g., Network Calculus [64] or Trajectory Approach [73]). Moreover, the AFDX switches contain forwarding policies based on the VL concepts which are statically defined. This feature eliminates online reconfiguration in the AFDX architectures.

2.4.4 Ethernet AVB

Audio Video Bridging (AVB) [10, 11, 12], as a set of technical standards developed by IEEE, is mainly designed to be used in the automotive industry. Ethernet AVB supports clock synchronization, bandwidth reservation and traffic shaping services. In the AVB standards all messages must have priorities and messages can share a priority level. A set of traffic with the same priority belong to a same traffic class. Moreover, within each traffic class the messages are treated following the FIFO policy. Two traffic classes, known as class A and B, are defined in the standard, where class A has higher priority than class B. A Credit-Based Shaping Algorithm (CBSA) for each of these two classes is specified to handle the traffic in classes A and B. The traffic in class A (similarly in class B) can be forwarded when the credit is zero or positive. During the transmission the credit is reduced with a defined rate called *sendSlope*. The credit is replenished at a particular rate which is called *idleSlope*. The credit is replenished in two scenarios. First, if there is no message in the corresponding traffic class in the queue, and second, when a message is waiting for transmission and there is another interfering message being transmitted. Note that when the credit is positive and there is no message in the queue to forward, the credit becomes zero immediately. The non-real-time traffic is transmitted when there is no message in classes A and B waiting, or when the credit for them is negative.

The real-time performance of IEEE AVB has been investigated extensively in multiple application domains, namely automotive, aeronautics, and industrial automation. For automotive networks, AVB is one of the possible candidates for real-time communication [90, 91]. Moreover, the AVB suitability for supporting traffic flows of both Advanced Driver Assistance Systems (ADAS) and multimedia/infotainment systems was investigated [92, 93, 94]. In addition, AVB is able to deal with real-time traffic in industrial automation [95, 96]. However, it turned out that further improvements for AVB are needed for its

use in industrial automation. In order to tackle the improvements several types of traffic shapers are analyzed, where Time-Aware Shaper (TAS) proved to be a solution with lowest latency along with a good jitter performance, albeit with an increased configuration cost [97]. Therefore, there are variations of the Ethernet AVB being proposed to provide a support for time-sensitive traffic. For instance, a higher priority class is introduced such that the traffic of that class does not go through the CBSA shaper [13]. This type of traffic is called Scheduled Traffic (ST) and is having its transmissions planned offline.

2.4.5 EDF scheduled switching

The EDF scheduled switching [98] is an early technique based on channel reservation mechanisms supported by enhanced switches. The system architecture contains a real-time layer on the switch and end nodes. This layer performs the required tasks including real-time channel establishment, admission control, time synchronization and forwarding of real-time traffic. In order to transmit a real-time message, a real-time channel should be established. The channel establishment is done using a request and answer mechanism. The node, which has a real-time message ready to be sent, issues a request to the switch containing the message information such as its period, deadline and transmission time. A connection ID is used to distinguish between several parallel requests. The switch checks the feasibility of establishing the real-time channel by verifying that the real-time guarantee for the message can be achieved. Then, the switch sends the request to the destination node, if the establishment in the first phase was acceptable. The destination node also analyzes the request and sends back the answer to the switch where the necessary actions to update the system takes place. The switch and nodes contain two queues per port, a FIFO type queue and a deadline-sorted queue. The former is used for non-real-time messages, whereas the latter is used for real-time messages. For the queues in both switch and nodes, the EDF scheduling algorithm is implemented. As the model is presented for a single-switch architecture, the deadline of a message is partitioned into two relative deadlines for the connections from the source node to the switch and from the switch to the destination node. For this purpose, a deadline partitioning function is proposed to create relative deadlines for all messages. A feasibility study based on utilization of messages according to Liu and Layland [57] analysis is presented for this protocol.

2.4.6 EtheReal

Similar to the EDF scheduled switching, EtheReal [99] is another early technique based on channel reservation along with enhanced Ethernet switches. This protocol supports real-time and non-real-time traffic. Two services, one to support the real-time traffic and another for non-real-time traffic, are implemented in the switch. In this proposal there is no need to modify the operating system in the end nodes. The real-time service provides a reserved bandwidth by means of a connection setup to send the traffic with a minimum delay. The non-real-time service is developed specifically for legacy traffic without providing guarantees.

The connection setup is done by sending a reservation request to a user-level process called Real-Time Communication Daemon (RTCD). The reservation request contains the respective QoS requirements including average traffic rate and maximum burst length.

2.4.7 PROFINET IRT

PROFINET IRT [100, 101, 5] is developed based on modified Ethernet switches and it is part of the IEC 61158 standard. The protocol supports different classes of services according to the applications. These classes include RT class UDP, RT class 1 to RT class 3 and non-real-time (NRT) class which has lower priority than RT class UDP. RT class 3 has the highest priority and it is forwarded according to a static communication schedule. The communication is cyclic-based and each cycle is divided into individual communication intervals. The individual intervals are reserved for the classes and each node must respect that using a precise clock synchronization. The traffic transmission is planned offline and loaded into the nodes. Other transmissions besides the planned ones are not allowed and in case of having those the switches block them.

2.4.8 EtherCAT

EtherCAT [7] is a protocol based on the master-slave technique, which was developed originally by Beckhoff. Typically, there are several slave nodes with one master node in an EtherCAT network. There are various master implementations for the master node in different operating systems. Moreover, slave nodes use custom available hardware, such as FPGA and ASICs solutions. Telegrams are generated and sent through the network by the master node crossing all the slave nodes and sending back to the master node. The

main idea of EtherCAT is that the slave nodes can read and write the telegram on the fly, as it passes through them. This generates a small delay in read and write processes of the nodes, making a short latency for the telegram circulating the network. Normally, a large number of slave nodes use only one telegram in the network. A performance study of EtherCAT and a comparison with PROFINET IRT is done by Prytz [102].

2.4.9 Ethernet POWERLINK

Beside the solutions based on enhanced Ethernet switches, there are also solutions that are proposed based on overlay protocols to control the traffic submitted to the COTS switches. One of these solutions is Ethernet POWERLINK [6] that uses a master-slave technique. The protocol supports both periodic and aperiodic messages using two isolated phases in cyclic time slots. Each cycle is composed of four phases including Start phase, Isochronous phase, Asynchronous phase and Idle phase. In the Start phase, the master node sends a synchronization message to notify the slave nodes about the starting of a new cycle. The master then polls the periodic messages during the Isochronous phase by sending a polling request to each of the messages. Progressively, the slave nodes reply by a polling response that is broadcasted to all nodes. Within the Asynchronous phase, the master sends a particular message, named Start of Asynchronous (SoA) to a specific node. Then, the node replies back with the asynchronous message. At the end of the cycle, an idle phase is considered to enforce a precise cycle start.

2.4.10 The FTT-SE protocol

There are different techniques to provide real-time communication over COTS Ethernet switches, mostly along with software modifications in the nodes. One of these techniques is the master-slave technique that caters features such as efficient scheduling policies and QoS management. The Flexible Time-Triggered Switched Ethernet (FTT-SE) protocol [103, 15] is one of the mentioned protocols in which a particular node takes the master role and coordinates transmission of traffic among the slave nodes. The communication occurs within time-slots, which are called Elementary Cycle (EC). The master schedules the ready traffic online considering the time available in each EC. The scheduling method can be based on fixed-priority or EDF algorithms. The scheduling information is broadcasted by the master to the slave nodes at the beginning of each EC by a particular message called Trigger Message (TM). The slave

nodes decode the information and initiate transmission of scheduled traffic. The protocol handles real-time periodic, real-time sporadic and non-real-time messages, each with a dedicated transmission window in the EC.

2.4.11 The HaRTES architecture

The Hard Real-Time Ethernet Switching (HaRTES) architecture [104, 63] uses the same concept as the FTT-SE protocol, i.e., the FTT paradigm, albeit using a modified Ethernet switch. Basically, the master module is implemented inside the switch instead of allocating a particular node for it. Therefore, the switch is responsible to schedule the traffic, both periodic and sporadic, online according to any desired scheduling policy. The HaRTES switch implements a central repository which contains all information related to the traffic management, namely the message attributes. Then, in each EC the switch indicates the new activations and schedules them for the EC by checking whether the messages fit within the window in the EC. The transmission is triggered by the TM at the beginning of each EC. The switch does not have activation information of real-time sporadic messages. Therefore, these messages are transmitted without being triggered by the TM, within their respective windows, hence being shaped by the switch.

2.5 MTU for Ethernet frames

A prominent feature of Ethernet-based protocols is that a packet can hold data having a size up to 1500 bytes. This is relatively large compared to other communication protocols such as CAN with a limited size of 8 bytes for data per packet. The maximum data size that a packet can carry in Ethernet can be configured by the Maximum Transmission Unit (MTU). It should be noted that configuring the MTU size in the network may have a big impact on the performance of the protocol. For instance, limiting the MTU size of the packets in a specific in-vehicle network based on standard Ethernet can reduce the latency of the packets significantly [105].

The benefits and drawbacks of data fragmentation in large scale heterogeneous networks have been discussed in [106]. Sending large packets of data through the network results in an increase of throughput as the relative protocol overhead is reduced. However, in heterogeneous networks the MTU sizes can vary when a message is transmitted over different routes. Therefore, the large packets may need to be fragmented into smaller packets. In order to deal with

this problem, optimal routing techniques are proposed to reduce the fragmentation as much as possible. Moreover, several fragmentation techniques [107] are proposed to obtain higher performance.

In energy-constrained wireless networks, such as a Wireless Sensor Network (WSN) or an ad-hoc network, the packet length plays a very important role. Using large packet length reduces the energy consumption, whereas it increases the latency. Therefore, configuring the packet size during the design phase of such systems becomes a challenge. There are several works that are addressing this challenge keeping different targets for choosing the best packet size. In the work presented by Chen *et al.* [108] and Sankarasubramaniam *et al.* [109], such a problem is discussed and the effect of the protocol header by fragmentation of packets on the average performance is investigated. Moreover, in the work presented by Chudasama *et al.* [110] and Vuran *et al.* [111] the optimum packet size is studied based on three objectives, including throughput of the packet, consumption of energy, and utilization of resources.

In addition, there are several works focusing on specific applications to find an optimal data packet size. For instance, optimum packet size in underwater wireless networks is studied in [112]. In addition, the effect of the MTU size in the performance of wireless video is investigated in [113], in which the goal is to increase the quality of the video by increasing the signal-to-noise ratio. The work presented by Lin *et al.* [114] proposed a solution to find the optimum MTU size in power line communication networks. In healthcare monitoring systems with various error conditions the packet size setting can alleviate congestion problems [115]. There are also some works done to adjust the packet size in wireless networks dynamically. For instance, the packet size is selected based on the requirements coming from security issues [116]. Moreover, in the work presented by Frantti *et al.* [117] the packet size is adjusted during runtime using PID and fuzzy closed loop control, targeting minimum delay for packets. Nevertheless, the above mentioned proposals are focused on wireless networks with different goals in mind, mostly looking at a specific application.

Similar problems exist when scheduling real-time tasks with limited preemptions [118]. A task can be split to several chunks with various sizes due to preemptions, which is similar to the fragmentation of a message into several packets. In a set of tasks, selecting the preemption points of a lower priority task changes the size of its chunk, hence it affects the blocking term for a higher priority task. Therefore, an algorithm to select the optimum preemption points of the tasks is given by Bertogna *et al.* [119], where the goal is to minimize the preemption overhead. Moreover, in the same context, the longest duration of the last chunk of the task is determined by an algorithm given by Bertogna *et*

al. [120], in order to make the system schedulable. However, the above mentioned problem has less complexity compared with the MTU configuration, as the effect is only from the lower priority to the higher priority tasks. The effect of the MTU size is not only from the lower priority messages, but also from the higher priority messages as well as from the idle time in cyclic-based communications.

Lee *et al.* [105] focused on in-vehicle networks based on the standard switched Ethernet. The authors proposed a solution to limit the path MTU of lower priority packets that share routes with safety-critical packets in order to decrease the latency of the packets. However, the paper focuses on the path MTU, which is the smallest MTU of the path from a source to a destination. Moreover, Behnam *et al.* [121] proposed two algorithms (optimum and simplified) to find one optimum MTU for all messages in the scope of the single-switch FTT-SE protocol. Nevertheless, the above algorithms are applicable only on small networks consisting of a single Ethernet switch, and they are not scalable. Moreover, the algorithms in [121] use utilization-based schedulability tests, and the algorithm in [122] uses a tighter schedulability analysis based on an approximation of the response time analysis. Therefore, in both cases they result in less accurate solutions.

2.6 Dynamic reconfiguration

Distributed embedded systems are being developed for different applications, including mission-critical and long-running systems. In many of these applications, the availability requirement of the system imposes restrictions on restarting or taking them offline. For instance, for a system it may not be acceptable to make major interruptions due to safety or economical reasons. This means that such systems must be highly available and adaptable. Therefore, dynamic reconfiguration protocols are developed to allow such systems to evolve during run-time [123]. Changing parameters or even parts of a software system has been addressed in the literature for a long time, e.g., [123, 124]. However, these works did not highlight the timeliness guarantee during the change.

There are some works targeting reconfiguration in processors. In these works the intention is adding, removing or changing currently running real-time tasks, an operation which is known as a mode change. Several studies have been done to control the tasks running in processors to reduce the speed or offload some tasks in case of increased temperature, e.g., [125]. Santambrogio *et al.* [126] implemented an online reconfiguration mechanism in a specific

operating system aiming for the performance enhancement. A survey is presented by Real and Crespo [127] covering mode change protocols in processor domains. Moreover, a recent work is presented by Kumar *et al.* [128], where servers in the processor are assumed as resource reservation.

In the context of distributed systems, dynamic reconfiguration is typically handled in a middleware layer. Garcia *et al.* [129] presented a dynamically reconfigurable service-oriented real-time middleware, while in another work by the same author [130] a software framework for reconfigurable multimedia applications is proposed. However, both works follow a soft real-time approach that tolerates a few deadline misses. Brinkschulte *et al.* [131] presented a middleware for real-time distributed systems keeping the timeliness guarantee during the reconfiguration phase. The main idea is to change the service versions, add or change the software when the system is running. The middleware is designed over the OSA+ platform, which is an architecture used in embedded real-time systems. Moreover, a scheduling algorithm is proposed by Wisniewski *et al.* [132] in order to bring flexibility to time-triggered Ethernet based networks. In the same network a reconfiguration protocol is proposed [133]. A middleware is proposed by Marau *et al.* [134] to provide dynamic reconfiguration in FTT-SE networks [15]. In order to achieve the reconfiguration, a linear time-complexity admission control is proposed [135]. The above mentioned protocols cannot be directly applied within the multi-hop HaRTES architecture and the multi-hop FTT-SE protocol, simply because they are optimized for specific scheduling methods. Chapter 7 presents reconfiguration protocols which are compatible with the scheduling methods and the schedulability tests for the mentioned networks.

2.7 Resource reservation

Software in real-time systems has become overly complex due to advances in computer-controlled functions. One way to deal with such complexity is to split the system into several sub-systems. The sub-systems are then allowed to be developed by different teams independently. However, the sub-systems still share the system resources in terms of processors and network bandwidth. In order to support isolation and independence among the subsystems after their development, resource reservation techniques are used. The timing requirements of each sub-system can be verified independently. Then, the sub-systems can be integrated together without a need for analyzing the complete system. This section describes methods for reserving resources in processors, networks and distributed embedded systems.

2.7.1 Reservation in the processor domain

There are many resource reservation techniques in the processor domain such as virtual machines [136] and reservation support for time-sensitive multimedia applications [137]. As it is explained before, shared-driven scheduling is an alternative scheduling method in which different sub-systems can be scheduled in isolation and with different policies. This method of scheduling, i.e., using server-based scheduling, helps in reserving resources for each sub-system. In this scope, several proposals are presented in the literature based on different types of servers, such as using deferrable servers [38], sporadic servers [37] and hierarchical partitions [42]. In the above mentioned works, the proposals are complemented with a response time analysis. In order to present a tight analysis, many methods are adopted. Lipari *et al.* [43] proposed an analysis for periodic servers. The analysis for hierarchical partitioned scheduling is presented by Shin *et al.* [138], which considers a generic periodic server model for both fixed-priorities and EDF scheduling algorithms. The analysis presented by Almeida and Pedreiras [139] uses the same concept as in [138], however it covers a more realistic task model with release jitter.

2.7.2 Reservation in the network domain

Generally, the same concept as used in the processor domain is also used for resource management in networks. A general category of resource management in networks is traffic shapers [140]. Shapers limit the amount of load submitted to the network by using different policies such as credit-based shaping, e.g., as done by Ethernet AVB networks. Moreover, some real-time Ethernet protocols enforce a cyclic-based transmission in which each cycle is divided into several transmission phases. This way, a transmission window is allocated to a set of messages, resembling server-based scheduling inherent in the processor domain. In addition, the hierarchical scheduling model has been used in the network domain, such as the hierarchical scheduling framework implemented by the FTT-SE protocol [141] and the HaRTES architecture [63].

2.7.3 Reservation in distributed system domain

In the context of distributed embedded systems, very few works have addressed the reservation of resources, where the resources include both processors and networks. Some of these works consider soft real-time systems targeting multimedia applications, such as [142] and [143]. An adaptive QoS control is

developed by Cucinotta *et al.* [144] to control the reservation during run-time. An alternative framework is proposed by Khalilzad *et al.* [145] that supports real-time components containing several tasks and messages. However, the above frameworks are presented for soft real-time systems.

A distributed kernel framework is developed by Lakshmanan *et al.* [146]. It guarantees the satisfaction of end-to-end timing requirements in distributed real-time applications. The application model is based on a distributed task graph, where each application is divided into several sub-tasks that communicate with each other. Each sub-task is allocated to one processor and the end-to-end deadline is decomposed to local deadlines. The reservation on the network is achieved by traffic shaping. A general model, called Q-RAM [147], provides sharing of resources among multiple applications by controlling QoS in the system. The Q-RAM model has been extended to cope with the systems with rapid changes [148]. However, the main focus in these works is on distributed tasks while the network protocols have not received much attention.

A global resource management model, called D-RES [149], supports reservations on both nodes and networks with end-to-end timing guarantees. In this model a sub-system contains tasks and message, albeit with one task executing in each node. Chapter 8 presents a distributed resource reservation model, which seemingly resembles the model in D-RES. Nevertheless, the model in this work allows a sub-system to contain multiple distributed transactions each of which consists of several tasks executing in a single node. Moreover, the model supports various activation patterns for transactions such as trigger, data and mixed chains.

Chapter 3

System Model

This chapter presents a complete system model which is used in the entire dissertation. The system model includes a detailed description of the single-switch FTT-SE protocol and the single-switch HaRTES architecture. In addition, a network model, a traffic model and a task model are presented in this chapter.

3.1 Protocol description

This section is composed by detailed descriptions of the architectures relevant to this dissertation, including the switch behavior and the scheduling methods.

3.1.1 The single-switch FTT-SE protocol

The Flexible Time-Triggered Switched Ethernet (FTT-SE) protocol [103, 15] uses the master-slave technique to coordinate all traffic in the network. This protocol supports real-time periodic, real-time sporadic and non-real-time traffic. The former is classified as *synchronous traffic*, which is time-triggered, and the two latter are categorized as *asynchronous traffic*, which is based on an event-triggered manner. An example of the FTT-SE architecture is depicted in Figure 3.1.

A protocol is developed to schedule and transmit the messages in the FTT-SE architecture. According to this protocol, the master node organizes the traffic transmission in fixed-duration time-slots called Elementary Cycle (EC). The data communication in each EC is divided into two specific windows to

handle synchronous and asynchronous traffic, which is called the synchronous window and asynchronous window, respectively. Figure 3.2 shows the EC and the communication windows in the FTT-SE protocol.

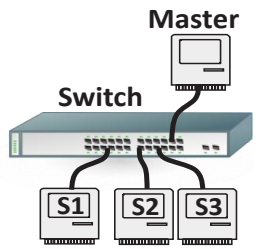

Figure 3.1: An FTT-SE network example.

The master node schedules the traffic online according to a user defined scheduling policy, for example the fixed-priority scheduling policy, considering the dedicated window within the EC. If a message fits in the transmission window, the master node encodes it into a particular message called Trigger Message (TM) to be broadcasted to the slave nodes at the beginning of the EC (Figure 3.2). Once the nodes receive the TM, they decode it and initiate the transmission of the scheduled messages. The nodes contain a database for the message parameters, called Node Requirements Data Base (NRDB), that allows determining the scheduled messages. The process of decoding and initiating the transmission take an amount of time that depends on the processing speed of the slave nodes. This amount of time is called turn around time (TRD), that is illustrated in Figure 3.2.

The asynchronous messages make use of a signaling mechanism [150] that allows the master to become aware of them and to consider them in its internal traffic scheduling. The signaling mechanism is based on the usage of a so-called signaling message (SIG) that is sent by the nodes to the master node, informing it of the status of the nodes' queues (for example, messages A and B in Figure 3.2). Whenever an asynchronous message becomes active in one node, this node informs the master in the next SIG message that it sends to the master, thus the master schedules the asynchronous message for the upcoming ECs.

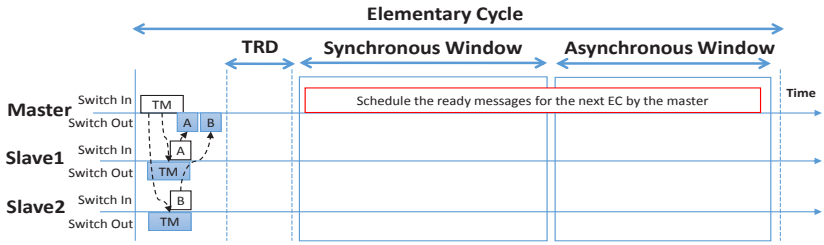

Figure 3.2: The EC in the FTT-SE protocol.

The main aim of the master node is to schedule the traffic online without causing overrun in the EC, i.e., the scheduled traffic must be received before the end of the EC. The master node contains a repository to keep the parameters of the messages. The database that keeps the data is called System Requirements Data Base (SRDB). The scheduler in the master node determines the activation of messages and inserts them into a ready queue. Then, the scheduler picks messages from the ready queue and checks whether they fit in their dedicated window in that EC. This procedure continues until the last message in the ready queue. The unscheduled messages are kept in the ready queue for the next ECs. To keep track of the utilization of the windows in each link connecting to the switch (assuming common full duplex switches), the master considers two bins per link and per window, one bin representing the uplink (node to switch) and the other bin the downlink (switch to node). Then, the scheduler starts from the higher priority messages and fills up the bins associated to the links in the message path while considering the delays imposed by the switching itself and the interference caused by the other traffic.

Let us assume an example that is depicted in Figure 3.1. Moreover, assume that two synchronous messages, m_1 and m_2, are ready to be transmitted from node S1 to node S2 and from node S3 to node S2, respectively. These two messages share a link to node S2. Figure 3.3 illustrates the bins corresponding to the uplink of S1 and S3, and the downlink of S2. Assuming m_1 has higher priority than m_2, the scheduler fills up first the uplink S1 bin with m_1 transmission time. It also fills up the downlink S2 bin with the message transmission time plus its switching delay resulting from crossing the switch. Then the scheduler checks m_2 and fills up the uplink of S3 and the downlink of S2, the same way for m_1. If both messages fit in the bin, the scheduler encodes them into the TM for broadcasting to the nodes. In case m_2 does not fit in one

of the respective bins, the scheduler keeps that message in the ready queue for the next EC scheduling.

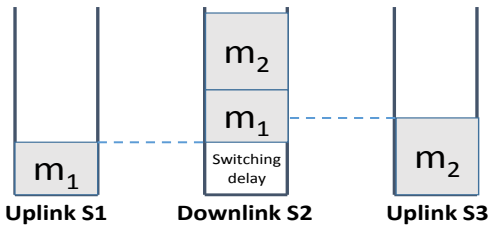

Figure 3.3: The allocated bins in the FTT-SE protocol.

3.1.2 The single-switch HaRTES architecture

The Hard Real-Time Ethernet Switching (HaRTES) switch [104, 63] is a modified Ethernet switch based on the master-slave technique where the master is implemented inside the switch. Figure 3.4 depicts the abstract functional structure of the switch. The Packet Classification module, implemented for each input port, is analyzing the incoming packets to distinguish the traffic types. Then, the module appends the traffic to the memory queue in the Memory Pool module. The Master module contains a scheduler, an admission control, a QoS management and a repository of the traffic attributes. These attributes include the deadline, minimum inter-arrival time or period, message length and priority.

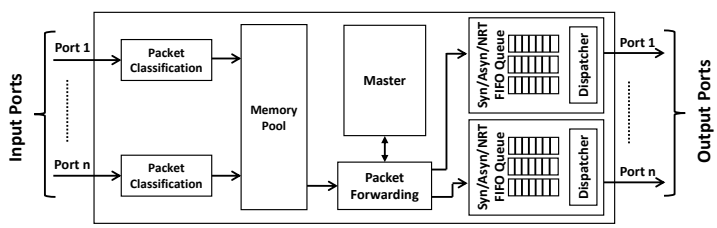

Figure 3.4: The HaRTES switch structure.

For each output port three FIFO queues are implemented to handle synchronous, asynchronous and non-real-time packets, which are identified as Syn, Asyn and NRT respectively in Figure 3.4. Finally, the Dispatcher module in the output port allows the traffic transmission within the associated windows. The Packet Forwarding module inquires the repository to determine a set of ports where the consumers of the packet are attached and it inserts the packets into the output FIFO queue(s) based on the packet type. 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 non-real-time traffic, yet with restricted bandwidth reserved for such traffic.

Similar to the FTT-SE architecture, the master module is responsible for scheduling the traffic within the EC. Moreover, the EC is divided between two windows, one for scheduling of synchronous traffic and the other one for asynchronous traffic. Controlling the transmission within these windows is performed by the Dispatcher module. In each EC, the switch determines new activation of the synchronous messages, it checks whether the message can be transmitted within the coming EC. The scheduled messages are encoded into the TM, to be transmitted to the nodes at the beginning of the EC. The slave nodes receive the TM, decode it and initiate the transmission within the associated window. Unlike the FTT-SE architecture, the asynchronous messages are transmitted to the switch autonomously without triggering by the TM. The switch buffers the asynchronous messages and schedules them for the next ECs. Note that the asynchronous messages are not allowed to be sent within the synchronous window. Thus, during the synchronous window, such messages are buffered in the switch. This way temporal isolation among the synchronous and asynchronous messages is achieved.

3.1.3 FTT-SE vs HaRTES

Despite the similarities between FTT-SE and HaRTES, there are also subtle but important differences, which have a strong impact in the operation and performance of both architectures. In particular, the master module is inserted in the HaRTES switch to overcome some limitations such as capability of temporal control by the switch. In the FTT-SE architecture, the nodes should be FTT-compliant to respect the timing of the EC. This requires a specific network driver and legacy nodes may not respect the protocol timing. By adding the master inside the switch the traffic confinement is achieved even for the legacy nodes. Moreover, due to having a master module inside the HaRTES switch,

some unauthorized real-time traffic can be blocked to prevent the interference to the other traffic. Moreover, non FTT-compliant nodes can be connected to the HaRTES switch without jeopardizing the real-time services. This is in fact due to having traffic confinement in the HaRTES switch.

In the HaRTES switch, the asynchronous messages are transmitted autonomously without being triggered by the master module, while in the FTT-SE architecture the asynchronous messages are polled by the master node. This eliminates the signaling mechanism in the HaRTES architecture, hence reducing the scheduling algorithm complexity.

Another difference is regarding the reserved bandwidth in different links. In the HaRTES architecture it is possible to have different bandwidth in different links, according to the actual load. This property results in more efficient bandwidth usage. Furthermore, unlike the COTS switches, HaRTES has ability to buffer the traffic due to implementation of a memory pool in the switch. Therefore, it is possible to buffer the traffic for a few ECs before transmitting to another switch or node. Beside the advantages of the HaRTES architecture, the FTT-SE architecture uses COTS Ethernet switches. Due to the wide availability of standard switches, the FTT-SE architecture provides a low cost solution compared to the HaRTES architecture. The HaRTES switch is developed using an FPGA-based platform which makes it somewhat complex in implementation.

3.2 Network model

In the FTT-SE architecture the switch is a standard switch, while in the HaRTES architecture the switch is a modified Ethernet switch. However, the model for both switches is the same in the sense that both are of store-and-forward type and provide full-duplex where the input and output of a switch port are isolated. Therefore, the receiving message does not delay the transmission of a message. A *link* is defined as a unidirectional connection between a node and a switch, as well as a connection between two switches. This means that each physical connection in the network is composed by two links, one per each direction. The link is denoted by l. Moreover, switches impose delays to the messages crossing through them, which are categorized into two components. The first switching delay component is *hardware fabric latency* which corresponds to the message processing time, whereas the second switching delay component is *store-and-forward delay* which corresponds to the time required to receive a message before forwarding it to the output link. The store-and-forward de-

lay depends on the message size and bit rate. The store-and-forward delay is denoted by SWD, while the fabric latency is denoted by ϵ.

3.3 Traffic model

For the traffic in this work, the real-time periodic model to represent both synchronous and asynchronous messages is used. A set of messages Λ, composed by N messages, is characterized as follows:

$$\Lambda = \{m_i(C_i, PK_i, D_i, T_i, P_i, \mathcal{L}_i), i = 1...N\} \tag{3.1}$$

In the networks, a large message is fragmented into several smaller packets. Therefore, C_i is the total transmission time of the message including its packets and overheads, such as inter-frame gaps and overhead of Ethernet frame. PK_i is the maximum packet size among the packets that compose m_i. Moreover, D_i is the relative deadline and T_i is the period of m_i. The model is assumed to be constrained, which means $D_i \leq T_i$. The period and deadline for the messages are expressed as an integer number of ECs in the FTT-SE and HaRTES architectures. For asynchronous messages T_i is the minimum inter-arrival time. The analysis presented in this work is restricted to unicast streams, hence only one destination port per message is considered. The multicast and broadcast streams can be handled by transforming them into multiple unicast streams and compute the response time of them separately. Then, the maximum response time among them is the worst-case response time of the multicast or broadcast message. P_i denotes the priority of the message. In this model, messages may share a priority level. In order to show a set of messages with priority higher than that of m_i, $hp(m_i)$ is used. Moreover, the set of higher and equal priority messages is given by $hep(m_i)$. Continuously, $lp(m_i)$ is used to show a set of messages with priority lower than that of m_i. In addition, $\mathcal{L}_i$ is the set of links that m_i crosses through. Each element in $\mathcal{L}_i$ presents a link l_x, and the number of links in $\mathcal{L}_i$ is denoted by n_i, which is extracted by $n_i = |\mathcal{L}_i|$. The set of links in the route of m_i is presented as follows:

$$\mathcal{L}_i = \{l_x | x = 1...n_i\} \tag{3.2}$$

A set of links which m_i passes from a specific link l_a until another specific link l_b in its route in the network is defined in Equation 3.3, where $\mathcal{L}_{i,a,b} \subseteq \mathcal{L}_i$ and $1 \leq a \leq b \leq n_i$.

$$\mathcal{L}_{i,a,b} = \{l_h | h = a...b\} \tag{3.3}$$

The scheduling policy for the messages in this work is considered to be the fixed-priority scheduling, where priority of messages is assigned according to the RM algorithm. Therefore, messages with the same period have the same priority.

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, the response time of a message m_i crossing between link l_a and l_b is specified by $RT_{i,a,b}$. Such a response time is the time duration between when the message is inserted to the queue in the switch/node with the output link l_a and the time the message is inserted to the queue in the switch/node with input link l_b. It should be noted that both the total response time and the response time between two particular links are expressed in number of ECs for the FTT-SE and HaRTES architectures.

All messages that are scheduled to be transmitted in one EC, should have been received by the end of the EC. In order to prevent overruns, scheduling of messages that cannot be fully transmitted within the transmission window is delayed for the next EC, for example, m_3 in Figure 3.5. This behavior introduces an idle time in each transmission window. The idle time is denoted by Id.

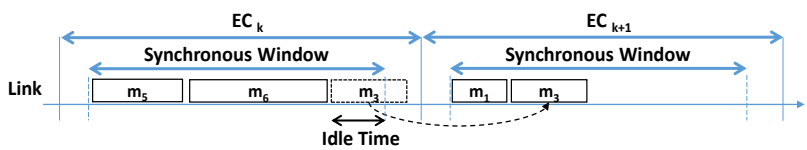

Figure 3.5: Inserted idle time demonstration.

3.4 Task model

Similarly to the message model, the task model in this dissertation is based on the periodic model. Each task is characterized by the following tuple.

$$\tau_i =< C_i, T_i, D_i, P_i, J_i > \tag{3.4}$$

The worst-case execution time of the task is denoted by C_i. A task can be activated periodically or sporadically. In the case of periodic activation, T_i

represents its activation period. Whereas, in the case of sporadic activation, T_i represents the minimum inter-arrival time between any two consecutive activations of the task. The relative deadline for the task is also presented by D_i. The term P_i and J_i in Equation 3.4 represent the priority and maximum release jitter of the task, respectively.

Chapter 4

Multi-Hop FTT-SE Architecture

One of the main intentions of the dissertation is to provide a solution for multi-hop communication when using the FTT-SE and HaRTES architectures. This chapter focuses on the former architecture, while the next chapter discusses the latter architecture. In order to extend the FTT-SE architecture, two solutions are presented. The first solution proposes an architecture with multiple master nodes to coordinate the transmission of messages. Whereas, the second solution proposes an architecture considering a master node per group of switches. In addition, for both architectures a response time analysis is presented for messages passing through the networks. Previously, an architecture for enabling a multi-hop communication was proposed by Marau *et al.* [19] that uses one master node connected to one of the switches. This architecture is designated the single-master architecture in which the master node is responsible for scheduling of all messages in the entire network. This chapter compares the proposed architectures with the single-master architecture. Therefore, before presenting the solutions, the single-master architecture will be presented.

4.1 Single-master FTT-SE architecture

The single-master architecture is depicted in Figure 4.1, in which one master node is attached to the top of the network hierarchy, controlling the traffic in the whole network. The master node is responsible to schedule messages

on-line according to any desired scheduling policy, on a cyclic basis. The EC is partitioned among two types of traffic, i.e., synchronous and asynchronous traffic, resulting in a design where the EC hosts one synchronous and one asynchronous window (Figure 4.2). The scheduler in the master node checks the synchronous messages, which are activated periodically, whether they can be transmitted during the associated window within the EC (i.e., within the synchronous window). The scheduled messages are encoded into a Trigger Message (TM) that is transmitted to all slave nodes at the beginning of the next EC. The TM transmission is based on the master-slave technique where the traffic transmission is controlled by a master node and triggering the transmission within specific cycles.

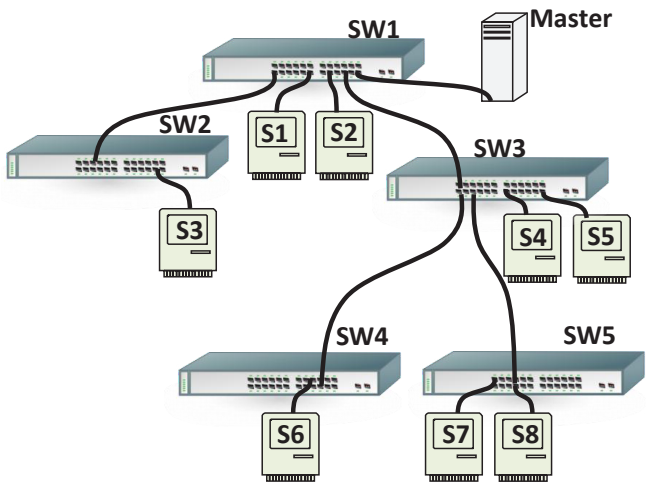

Figure 4.1: The single-master FTT-SE architecture.

The activation of asynchronous messages is unknown in advance and can occur at any time during the EC. Therefore, the same signaling mechanism in the single-switch FTT-SE architecture is used to inform the master node about potential pending requests. The information is sent via SIG messages, which are transmitted after reception of the TM (e.g., A, B and C in Figure 4.2). The master then schedules the asynchronous messages adequately and inserts them into one of the upcoming TMs. The slave nodes receive the TM, decode it and initiate the transmission of the scheduled messages and in parallel initiate

reception of data messages from other slave nodes. Note that, similar to the single-switch architecture, decoding the TM takes an amount of time (TRD in Figure 4.2). The actual length of TRD depends on the processing speed of the slave nodes and it is accumulated by increasing the number of hierarchy levels in the network. The response time analysis for the traffic transmitted through the switches in this architecture is developed and presented by Marau *et al.* [19].

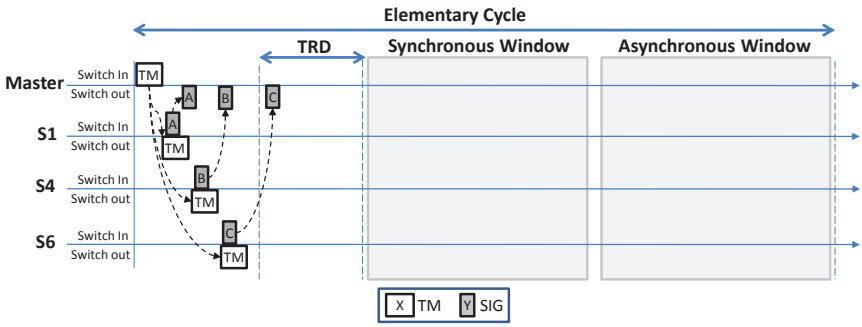

Figure 4.2: Elementary Cycle in the single-master FTT-SE architecture.

4.2 Multi-master FTT-SE architecture

This section proposes the architecture with multiple master nodes performing the traffic scheduling collaboratively.

4.2.1 The multi-master protocol

The multi-master FTT-SE network consists of multiple switches connected together in a tree topology, each with its own master, as illustrated in Figure 4.3. The switch in the top of the hierarchy is called the root switch. Moreover, a *sub-network* is defined as the ensemble of each switch along with all nodes that are directly connected to it. For instance, SW1, M1, S1 and S2 in Figure 4.3 is a sub-network. Each sub-network is a parent for the lower level sub-networks that are attached to it. The group of sub-networks that have the same parent sub-network is called a *cluster*. For example, Cluster 2 in Figure 4.3. The root sub-network can be included in its children cluster or it can be considered as

a separate cluster. However, in this work and for the sake of simplicity we assume that the root sub-network is included to the children cluster, as depicted in Figure 4.3.

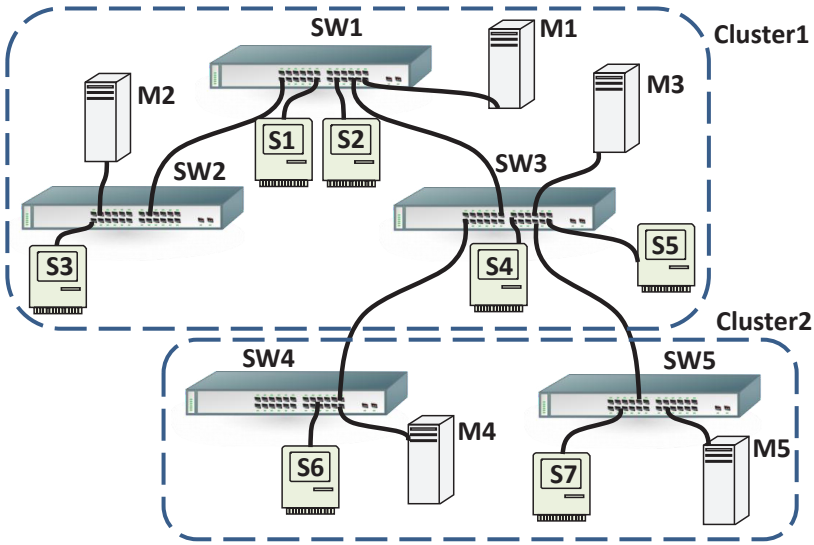

Figure 4.3: A multi-master FTT-SE network example.

In addition, the traffic in this architecture is divided into two categories. A message that is transmitted within a sub-network is called *local*, otherwise it is called *global*. As in any FTT protocol, the master schedules the messages on-line according to a scheduling policy. The scheduler is invoked periodically, exactly once every EC. Each EC, which is configured off-line, is mainly divided in two windows, one to manage the protocol called *Initialization Time* and another one to transmit the data messages which is called *Data Transmission Window* (see Figure 4.4). In the multi-master architecture, the data transmission window is further partitioned among all traffic types, including synchronous and asynchronous. Each of the windows is divided between local and global traffic. The global asynchronous window is further divided among the clusters, for example, Cluster1 and Cluster2 in Figure 4.4.

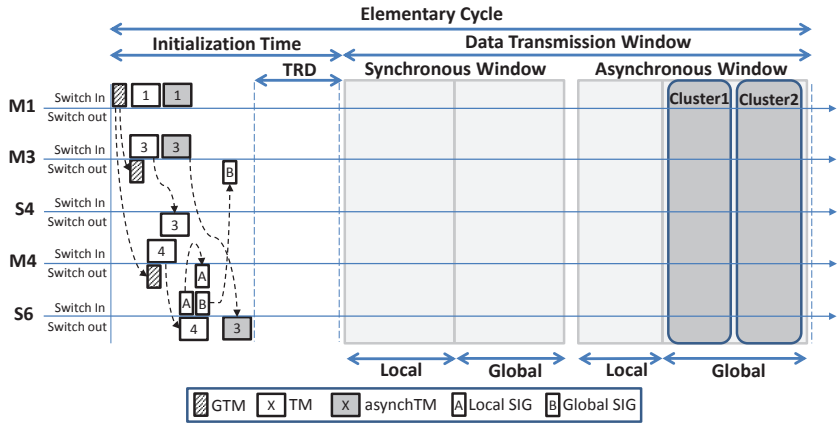

Figure 4.4: An EC in the multi-master FTT-SE architecture.

This architecture requires that the ECs of all master nodes are synchronized. In order to achieve this synchronization a particular message, called Global Trigger Message (GTM), is used. The GTM is broadcasted by the root master node and propagated down the entire network. The remaining master nodes synchronize upon receiving the GTM and start their local ECs. In the event that a GTM is lost, the receiving master node times out, triggers its own local EC and generates a GTM itself for the clusters below it in the topology tree. Certainly, other solutions are possible, such as using clock synchronization among master nodes, which however is considered out of scope of this dissertation.

Algorithm 1 describes the operation of a generic master node. The only different master is the root master, which follows a similar algorithm except that its cycle is timed, not synchronizing with the GTM. The main loop of the master nodes starts with broadcasting the GTM for the lower level masters (lines 7). Then the local EC is triggered, broadcasting the TM to all the nodes in its sub-network (line 8) and the asynchronous TM (asynchTM) to its children sub-networks (line 9). The TM conveys the IDs of the scheduled local/global synchronous and local asynchronous messages (for instance, TM4 from M4 to S6 in Figure 4.4). The global asynchronous messages, however, are scheduled by the parent master of each cluster based on the dedicated window allocated for that cluster (for instance, asynchTM3 from M3 to S6 in Figure 4.4).

Algorithm 1 Generic master node algorithm

```
1:  InitializeMaster()
2:  TM = initialTM
3:  asynchTM = initialAsynchTM
4:  receive(GTM) //sync with GTM
5:  timeOut = currentTime + EC + waitTime
6:  loop
7:      send(GTM)
8:      send(TM)
9:      send(asynchTM)
10:     //wait for all SIG messages
11:     waitSIG = currentTime + SIGwindow
12:     while currentTime < waitSIG do
13:     end while
14:     while inputBuffer(SIG*) do
15:         //read all received SIG messages
16:         receive(SIG*)
17:     end while
18:     update(LocSyncQ, LocAsyncQ)
19:     update(GlobSyncQ, GlobAsyncQ)
20:     //prepare TM and asynchTM for next EC
21:     TM = schedule(LocSyncQ, LocAsyncQ, GlobSyncQ)
22:     asynchTM = schedule(GlobAsyncQ)
23:     // sync with GTM for next cycle
24:     receive(GTM, timeOut)
25:     timeOut = timeOut + EC
26: end loop
```

Unlike synchronous messages, activation of asynchronous ones is unknown in advance and can occur at any time. A signaling mechanism [150] allows the slave nodes to notify the master of pending requests using SIG messages that are transmitted at the beginning of the EC. For instance, message A and B are SIG messages in Figure 4.4. Therefore, each master collects the local SIG messages of its sub-network and the global ones from its cluster (cycle starting in line 11) and updates the ready queues for all types of traffic (lines 18 and next). Then, the master node starts to pick the messages from the head of the queues until no more messages fit in the respective window and it computes the TM and asynchTM for the next EC (lines 21 and next). At the end of the cycle, the master node synchronizes again with the GTM to trigger the next cycle

(lines after 23). In case the GTM does not arrive in time, a time out occurs and triggers the cycle. The timeout is initialized with an offset $waitTime$ in order to keep track of a GTM that is receiving a timeout.

Note that the protocol entails several broadcast messages, namely the GTM, TM and asynchTM. Their dissemination, however, is confined by appropriate configuration of Virtual LANs (VLANs) so that they do not create undesired interference outside their domain. For example, each sub-network has an associated VLAN that confines the broadcast of the associated TM. The master nodes in a cluster and the parent master are also integrated in another VLAN within which the cluster parent master broadcasts the GTM and the asynchTM.

Algorithm 2 Slave node algorithm

```
1:  InitializeSlave()
2:  loop
3:      receive(TM) //wait on TM
4:      receive(asynchTM)
5:      send(SIG)
6:      (ID[1..m], m) = decode(TM, asynchTM)
7:      //send all DATA messages inTM and asynchTM
8:      for i = 1 → m do
9:          send(ID[i])
10:     end for
11:     //read all received DATA messages
12:     while inputBuffer(DATA*) do
13:         receive(DATA*)
14:     end while
15: end loop
```

The algorithm of the slave nodes is significantly simpler as shown in Algorithm 2. The main loop is synchronized on the reception of the TM and asynchTM (lines 3 and next). Concurrently (despite the sequential representation for convenience), the slave nodes send their SIG messages to their master node (line 5). Then, they decode the TM and asynchTM to determine which data messages (ID) they must transmit in the current EC (line 6). This process takes a time that depends on the processing speed of the slave node (TRD in Figure 4.4). Afterwards, the slave nodes initiate the transmission of the scheduled data messages (lines 8 and next) and the reception of the data messages sent by others (lines 12 and next). These two activities also happen concurrently, despite the sequential representation.

It should be noted that the windows inside the EC (Figure 4.4) are considered for traffic scheduling purposes, only, particularly to bound the amount of traffic of each type that can be transmitted every EC. At the time of transmitting the data messages scheduled for a given EC, they are all sent as soon as possible, as shown in the *send* cycle in Algorithm 2, line 8 and following.

4.2.2 Scheduling algorithm

The main aim of the master nodes is to schedule traffic on-line without causing overrun in the EC, i.e., the scheduled traffic must be received before the end of the EC. This is important to enforce the master traffic scheduling policy without interference of the management policy used by the switch in its queues. The scheduler in a master node picks the messages from the ready queues of all traffic types and checks whether they fit in their dedicated window in that EC. This procedure continues until the last message in the ready queues. The unscheduled messages are kept in the ready queues for the next ECs. Note that the master node considers an idle time in each transmission window to prevent overrun of messages in the allocated window. To keep track of the utilization of the windows in each link connecting to the switch, the master considers two bins per link and per window, one bin representing the uplink (node to switch) and the other bin the downlink (switch to node). Then, the scheduler starts from the higher priority messages and fills up the bins associated to the links in the message path while considering the delays imposed by the switching itself and the interference caused by other traffic.

Again considering the example in Figure 4.3, assume that two synchronous messages, m_1 and m_2, are ready to be transmitted from S1 to S4 and S2 to S4, respectively. These two messages cross several links and share the link between switches SW1 and SW3. Figure 4.5 shows the bin corresponding to the downlink of S4. Assuming m_1 has higher priority, the scheduler fills up first the bin with m_1 transmission time plus its switching delay resulting from crossing SW1 and SW3 (Figure 4.5.a). Then, the scheduler checks m_2, realizing that its switching delay is shorter than that of m_1 (Figure 4.5.b). Note that the switching delays affect messages transmitted in sequence only once, in fact, the longest such delay. Thus, there is no need to add the switching delay of m_2 in the bin (Figure 4.5.c). If the switching delay of m_2 was larger than the one for m_1, the scheduler would account m_2 switching delay instead (Figure 4.5.d). In general, in each EC the scheduler picks the maximum switching delay among the messages that fit in that EC. This mechanism will be used to develop the response-time analysis presented later in this chapter.

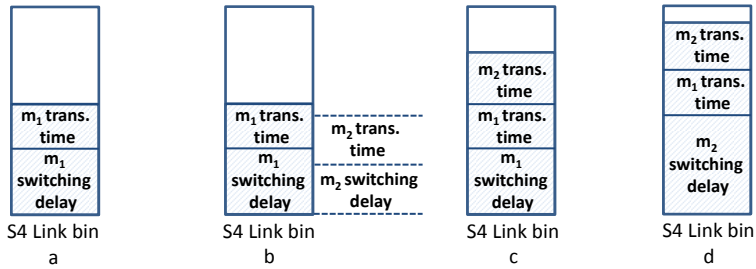

Figure 4.5: The bin representation.

4.3 Hybrid FTT-SE architecture

The single-master architecture uses only one master node in the architecture to coordinate the transmission of messages. Therefore, the master node should be powerful in particular for large networks with a high amount of messages. On the other hand, the multi-master architecture uses one master node per group of switches, which is beneficial for large networks. However, the complexity of cooperating the master nodes is high. Therefore, an alternative approach is to combine the two architectures that may have a potential to give better results. This can be done by connecting sets of single-master networks in *clusters*. An example of such an architecture is depicted in Figure 4.6. In this architecture, the traffic in each cluster is coordinated by a master node that is connected to the parent switch of the cluster. For example, M1 for Cluster1 and M3 for Cluster2 in Figure 4.6.

Based on the hybrid architecture, the traffic is classified in two categories according to the source and destination nodes of each message. If the source and destination of the message belong to the same cluster, the message is called *internal*, otherwise the message is called *external*. The window for data transmission is divided into internal and external sub-windows within synchronous and asynchronous windows as illustrated in Figure 4.7. Each master node schedules the internal messages considering the internal sub-window, while all master nodes schedule all external traffic in the network within the external sub-window in parallel. Thus, each master node ensures that enough external sub-window is available for its external messages. Again, the synchronization of the ECs can be achieved using different solutions, however in this work the same GTM mechanism is used as we did in the multi-master architecture.

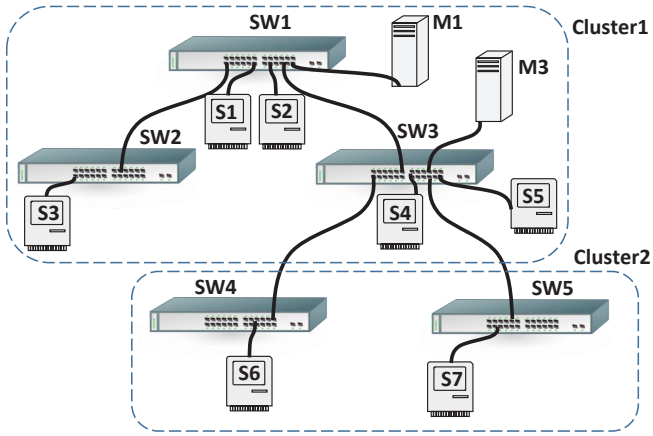

Figure 4.6: A hybrid FTT-SE network example.

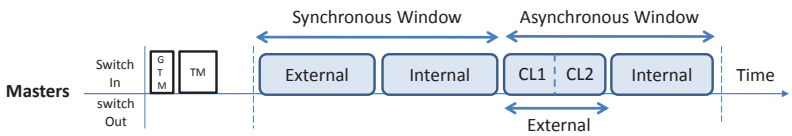

Figure 4.7: EC structure of hybrid architecture.

4.4 Response time analysis

In this section we compute the worst-case response time of messages in the multi-master and hybrid FTT-SE architectures. The FTT-SE scheduling is based on reserving a window for each type of messages that is provided periodically every EC. Within each reservation a scheduling policy is used to select the messages to be transmitted. This model is equivalent to the hierarchical scheduling model presented by Shin and Lee [138] and the associated analysis based on a *request bound function* (rbf) and a *supply bound function*

(sbf) is more suitable for such a network. Moreover, in this section we validate the proposed analysis and compare it with an analysis based on network calculus, which is presented by Ashjaei *et al.* [151] for the same architecture.

4.4.1 Worst-case response time analysis

The $rbf_i(t)$ represents the maximum load generated by m_i and all higher priority messages during the time interval $[0, t]$. It is computed by summing the transmission time of m_i with all delays that m_i may suffer during its transmission as shown in Equation 4.1, where Wl_i is the *Shared Link Delay* representing the maximum time that m_i is delayed by other messages in each link in its route from source node to destination node. Moreover, Wr_i denotes the time which m_i is delayed due to messages that do not share links with m_i, however that yet may interfere with m_i. This delay is called *Remote Link Delay* and will be explained later in this section. It should be noted that the store-and-forward delay is the maximum packet size among those that compose m_i. Thus, $SWD_i = PK_i$ in this analysis.

$$rbf_i(t) = C_i + (n_i - 1) \times (SWD_i + \epsilon) + Wl_i(t) + Wr_i(t) \tag{4.1}$$

Shared Link Delay. Message m_i, under analysis, may be delayed at each link in its route while crossing switches due to the interference from all higher priority messages that share those links with m_i. To avoid accounting for the same message multiple times, all messages that caused this delay in the previous links are excluded in the current link. Moreover, the store-and-forward and switch fabric latency of the messages generating the Shared Link Delay are collaborating to delay the message under analysis. The latter is due to the scheduling algorithm in which the scheduler fills up higher priority messages in the respective bin considering their store-and-forward delays, thus the message under analysis will be scheduled over those delays in that bin. Therefore, Wl_i is calculated in Equation 4.2, where $WT(m_i)$ is the set of messages with a type similar to that of m_i, which includes synchronous or asynchronous traffic as m_i.

$$Wl_i(t) = \sum_{\substack{\forall j \in [1,N], j \neq i \\ \wedge\ \mathcal{L}_j \cap \mathcal{L}_i \neq \emptyset \\ \wedge\ m_j \in hep(m_i) \\ \wedge\ m_j \in WT(m_i)}} \left\lceil \frac{t}{T_j} \right\rceil (C_j + (n_j - 1) \times (SWD_j + \epsilon)) \tag{4.2}$$

Remote Link Delay. The message that do not share links with m_i may still delay m_i indirectly through other messages. To show this effect, let us consider the following example, referring to the network that is shown in Figure 4.3. Four global messages are defined, including m_1 that is transmitted from S1 to S4, m_2 that is sent from S2 to S5, m_3 that is transmitted from S7 to S5 and m_4 which is sent from S6 to S4. Let us assume that m_1 has the lowest priority among the messages and $m_2 + m_3$ can be scheduled in one EC. Moreover, let us consider that m_1, m_2 and m_3 are activated simultaneously in the first EC and m_4 is activated in the second EC. In addition, m_3 is started to transmit with an offset in the source node due to other higher priority messages in the source node of m_3. The scheduling window for the first and second ECs is depicted in Figure 4.8. In this example, m_1 and m_2 have a shared link between SW1 and SW3 and due to the FIFO queue inside the switch it may be possible that m_1 is sent earlier in the FIFO even though it has lower priority (Link SW1-SW3 in Figure 4.8, EC1). Therefore, scheduling of m_1 can delay m_2 and in turn m_2 delays m_3 makes it overrun the window, although they do not share links. Thus, the scheduler postpones m_1 for the next EC to prevent any possible overrun.

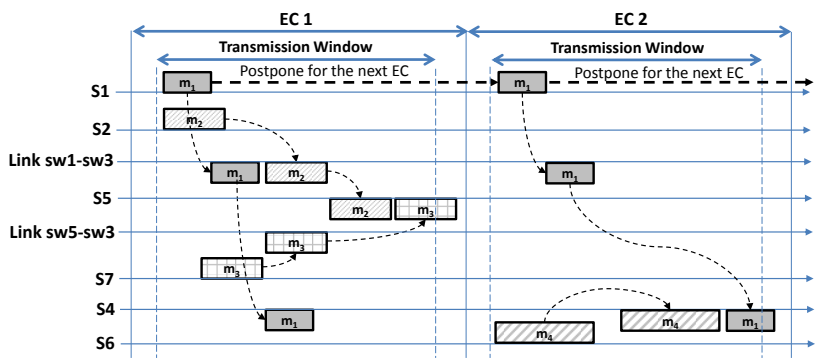

Figure 4.8: Window scheduling for remote link delay example.

In the second EC, m_4 is activated and m_1 was suspended from the first EC. However, as m_4 has a shared destination with m_1, and due to having a higher priority, m_4 is scheduled to be transmitted in the second EC, hence m_1 is again postponed for the third EC (considering enough size for m_4 to fill the

link of S4). If all messages have been released simultaneously then m_1 would be transmitted in the second EC as m_4 could be scheduled in the first EC.

In order to formulate the above outlined effect, in the worst-case response time analysis, all messages (m_k) that share links with those that are contributing to the Shared Link Delay (m_j) are taken into account. Thus, the Remote Link Delay is computed in Equation 4.3. Note that the messages which are already considered in the Shared Link Delay are excluded.

$$Wr_i(t) = \sum_{\substack{\forall k,j \in [1,N], k \neq j \neq i \\ \wedge\ \mathcal{L}_k \cap \mathcal{L}_j \neq \emptyset \wedge\ \mathcal{L}_k \cap \mathcal{L}_i = \emptyset \wedge\ \mathcal{L}_j \cap \mathcal{L}_i \neq \emptyset \\ \wedge\ m_k \in hep(m_j) \\ \wedge\ m_k \in WT(m_j)}} \left\lceil \frac{t}{T_k} \right\rceil C_k \tag{4.3}$$

The $sbf(t)$ is the minimum effective communication capacity that the network supplies in the time interval $[0, t]$. Note that in each EC, the utilization bandwidth provided for transmitting each type of message is $\frac{BW - Id_i}{EC}$, where BW is the specific time duration of the respective transmission window. The idle time is upper bounded by the maximum packet size allocated in BW, that is computed by Equation 4.4.

$$Id_i = \max_{\substack{\forall p \in [1,N] \\ \wedge\ m_p \in hep(m_i) \\ \wedge\ \mathcal{L}_p \cap \mathcal{L}_i \neq \emptyset \\ \wedge\ m_p \in WT(m_i)}} \{PK_p\} \tag{4.4}$$

Thus, $sbf_i(t)$ can be computed in Equation 4.5.

$$sbf_i(t) = \left(\frac{BW - Id_i}{EC} \right) \times t \tag{4.5}$$

The response time of m_i is computed based on $t^* = min(t > 0) : sbf_i(t) \geq rbf_i(t)$. Determining t^* requires checking the inequality at all instants in which $rbf_i(t)$ changes due to interference of other messages. Such a set containing check points is given by Equation 4.6.

$$\begin{aligned} &CP_{rbf_i} = [\cup cp_{m_q}, \forall_{m_q \in hep(m_i)} \wedge\ \mathcal{L}_q \cap \mathcal{L}_i \neq \emptyset] \cup \{T_i, C_i\}, \\ &where\ cp_{m_q} = \{T_q, 2T_q, ..., n_q T_q\}, n_q = \lfloor \tfrac{T_i}{T_q} \rfloor. \end{aligned} \tag{4.6}$$

The scheduling occurs every EC but in general we do not know exactly when a scheduled message will be transmitted within the EC. Therefore, we

compute the messages response times in number of ECs and the response time for m_i is given by Equation 4.7.

$$RT_i = \left\lceil \frac{t^*}{EC} \right\rceil \tag{4.7}$$

The analysis for the asynchronous traffic is essentially similar except for an extra delay caused by the signaling mechanism. In fact, an asynchronous request may have to wait maximum 1 EC before its node signals it in the next SIG message and the master then takes another EC to insert the respective asynchronous message in the ready queue. This extra delay is added to the RT_i for all asynchronous messages. For local traffic, messages may suffer from both Shared Link Delay and Remote Link Delay which are evaluated in Equations 4.2 and 4.3, respectively. However, local messages are transmitted through one switch, only, thus $n_j = 2$. Moreover, $rbf_i(t)$ for local messages is computed by Equation 4.1, with $n_i = 2$.

4.4.2 Improved response time analysis

One of the sources of pessimism in the presented response time analysis is the accumulation of the switching delay from all messages that contribute to the Shared Link Delay outlined in Equation 4.2. According to the scheduling algorithm presented in Section 4.2.2, during the scheduling process in each EC, the scheduler considers the maximum switching delay among all messages that are fit in each link. In contrast, in the analysis all switching delays are accumulated due to the lack of scheduling information at each specific time. This makes the analysis very pessimistic especially when the number of messages is large. For instance, assume that there are 30 messages that can be transmitted in 3 ECs. Therefore, 3 switching delays out of 30 are considered in the scheduling. However, in the analysis all 30 switching delays are added. In order to improve the analysis, Equation 4.2 is modified by separating the switching delay effect from the transmission time of messages in Equation 4.8.

$$Wl_i(t) = \sum_{\substack{\forall j \in [1,N], j \neq i \\ \wedge\ \mathcal{L}_j \cap \mathcal{L}_i \neq \emptyset \\ \wedge\ m_j \in hep(m_i) \\ \wedge\ m_j \in WT(m_i)}} \left\lceil \frac{t}{T_j} \right\rceil C_j + Is_i(t) \tag{4.8}$$

In Equation 4.8, the activation of all higher priority messages that can contribute to this equation can be extracted by knowing the current time t. We

define a multi-set $G_i(t)$ that contains the upper bound number of switching delays from each message that contributes to the Shared Link Delay at time t. A similar improvement has been done by Behnam *et al.* [152] in the scope of processor scheduling and response time analysis. This upper bound is equal to the maximum number of possible activations of messages within time interval $[0, t]$. Depending on the time t, a number of elements will be taken from $G_i(t)$ and to consider the worst-case scenario, the selected elements should be the largest ones. Therefore, we define a multi-set $G_i^{sort}(t)$ which contains the switching delay values from $G_i(t)$ sorted in a descending order. According to the scheduler, one switching delay per EC (the largest) is taken into account for scheduling. The number of ECs that have been passed in an interval $[0, t]$ is given in Equation 4.9. Therefore, in the response time analysis we select the first $z(t)$ number of elements from $G_i^{sort}(t)$ in order to consider the worst-case scenario for the switching delay effect in our analysis, as shown in Equation 4.10.

$$z(t) = \left\lceil \frac{t}{EC} \right\rceil \tag{4.9}$$

$$Is_i(t) = \sum_{l=1}^{z(t)} G_i^{sort}(t)[l] \tag{4.10}$$

It should be noted that if $z(t)$ is larger than the number of elements in the multi-set $G_i^{sort}(t)$, then the value of the extra elements is equal to zero. In addition, the presented improvement in the analysis can be applied for the case of the single-master FTT-SE architecture as well as the scheduling algorithm used by the master node is similar as the one used within the multi-master architecture.

4.4.3 Response time of messages in the hybrid architecture

A message in the network may suffer from delays and interference during the transmission. The interference is similar to the multi-master architecture, except that the windows allocation between the traffic types is different. The Shared Link Delay for message m_i occurs when any other higher priority message shares any links from the source to the destination node in the route of m_i. This delay is derived in Equation 4.2 considering that the messages should have the same window allocation as m_i in the transmission time ($WT(m_i)$). The Remote Link Delay is applicable as well, in which all the messages that have a

shared link with the Shared Link Delay messages set should also be taken into account. The Remote Link Delay is computed by Equation 4.3 considering the window allocation for the messages (WT). Finally, the $rbf_i(t)$ and $sbf_i(t)$ are calculated by Equations 4.1 and 4.5, in order to derive the response time.

4.4.4 Validation of the analysis

In this section we evaluate the presented analysis and compare it with the delay analysis based on Network Calculus proposed by Ashjaei *et al.* [151]. Moreover, we assess the level of pessimism embodied in both analysis methods compared with the simulation measurements in one particular example.

We consider a network containing 10 switches along with 30 nodes directly connected to the switches as shown in Figure 4.9. The network parameters are $EC = 10ms$, $TM = 12\mu s$, $SIG = 7\mu s$, $\Delta = 17\mu s$ and the capacity of the Ethernet network is considered to be $100Mbps$. Moreover, the transmission window during each EC is divided as follows. The synchronous local and global windows are selected to have values equal to $1.5ms$ and $2ms$, respectively, asynchronous local transmission window is $1.5ms$ and finally the asynchronous global transmission window is $4.4ms$. In this particular example, the network is composed of 4 clusters in which the asynchronous global transmission window is further split equally, i.e. $1.1ms$.

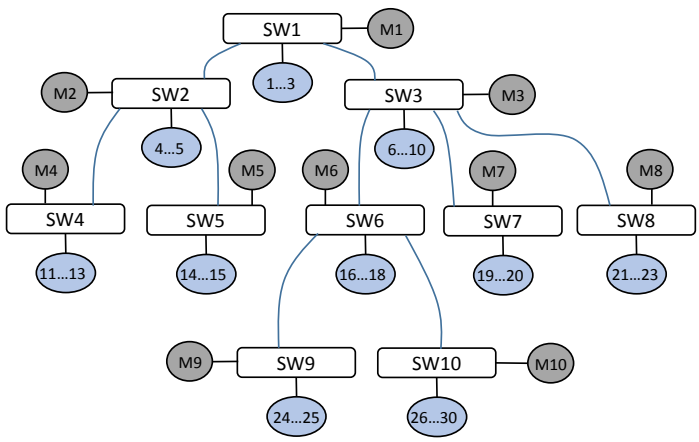

Figure 4.9: A network example (1...3 means node 1, 2 and 3).

We generate 90 messages that include all four types of traffic such that one specific message per each type is tagged to have a bad scenario among all other messages. In order to maximize the response time for global synchronous and asynchronous messages the priorities of the tagged messages are selected to be lowest and the route of the messages to be longest. Moreover, we delay the tagged messages with several higher priority messages. Furthermore, to generate a bad scenario for local synchronous and asynchronous tagged messages we set their priorities to the lowest and we delay them with several higher priority messages in the source and destination links. The parameters of the tagged messages are presented in Table 4.1.

Message id	Message Type	T(EC)	C(μs)
m_1	Global Synchronous	20	100
m_2	Global Asynchronous	18	150
m_3	Local Synchronous	20	150
m_4	Local Asynchronous	19	150

Table 4.1: Parameters of the tagged messages.

In order to simulate the outlined example we use a simulation tool, which is developed using the Matlab environment [153]. The messages which are generated to delay the tagged messages are considered to have different activation times in the simulation. We then measured the maximum response time of the tagged messages and compared it with the results obtained from the proposed response time analysis. This example does not guarantee to produce the worst-case situation since it is rather complex to determine that the worst-case is actually covered with accuracy, however the example is assuredly a very unfavorable case.

Figure 4.10 illustrates the maximum response time of the tagged messages measured over a 10,000 EC simulation time, the response time computed using the proposed analysis and the analysis based on Network Calculus. In the figure, the x-axis represents a message id and the y-axis shows the message delay in number of ECs.

Comparing the results of simulation, proposed analysis and Network Calculus analysis, the maximum measured delay from the simulation is always less than or equal to the outcomes of both analysis methods. We observed that the response time analysis gives tighter results due to the improvement compared with the Network Calculus analysis. An extreme level of pessimism provided by Network Calculus for global synchronous tagged message (m_1) is due to

the large number of interfering messages in the route of m_1 in which Network Calculus accumulates all their switching delay in the analysis. Although the presented analysis based on response time performs better, it still illustrates a level of pessimism varying between 50% for local synchronous tagged message (m_3) and 100% for the global asynchronous tagged message (m_2). The revealed level of pessimism for local and global asynchronous tagged messages is due to the adding of one extra EC in both analysis methods to capture the scenario in which the request for the asynchronous message needs to wait for the next EC. We also believe that another source of the pessimism is the accuracy of an actual worst-case scenario in the simulation and the effect of Remote Link Delay that might not be accurately captured in the simulation.

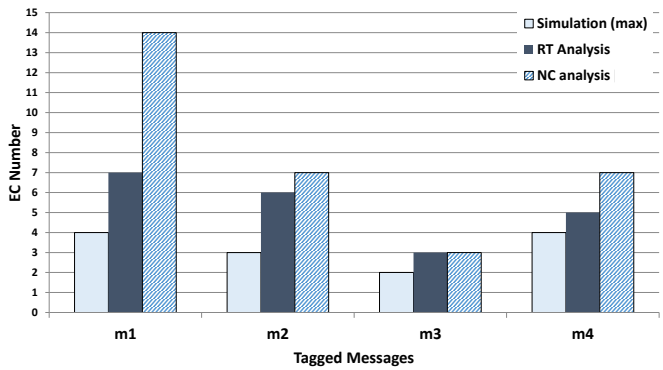

Figure 4.10: Response times of the tagged messages.

4.5 Comparative evaluation

This section compares the three architectures, being the single-master, multi-master and hybrid, quantitatively with respect to initialization time requirement, data transmission window and scheduler overhead.

4.5.1 Initialization time

Ideally, the initialization time should be a small fraction of the EC but it is lower bounded by the time needed to transmit and process the TM and SIG

messages. Here we compare the initialization time required by the architectures under similar scenarios.

Beyond their transmission times, the TM and SIG messages are delayed due to interference between different SIGs and to switch delays ($SWD + \epsilon$). Therefore, the data transmission window should start late enough to guarantee that TM and SIG messages have arrived before. The initialization time for the single-master architecture is given in Equation 4.11 based on the number of nodes in the network (N_{node}) and the turn around time (TRD).

$$\begin{aligned} init\_time = N_{dep} \times (C_{TM} + \epsilon) + max\{TRD, \\ N_{node} \times C_{SIG} + N_{dep} \times (C_{SIG} + \epsilon)\} \end{aligned} \tag{4.11}$$

C_{TM} and C_{SIG} are the transmission times of TM and SIG messages, respectively. N_{dep} represents the depth in the network topology, e.g., $N_{dep} = 3$ in Figure 4.3. Moreover, $N_{dep} \times (C_{TM} + \epsilon)$ is the TM reception instant in the farthest slave nodes. The transmission of SIGs overlaps with the TRD and thus we consider the maximum between the TRD and the time to transmit all the SIGs in the network, given by $N_{node} \times C_{SIG} + N_{dep} \times (C_{SIG} + \epsilon)$.

In the multi-master architecture, extra messages are used to control the protocol operation, including the GTM and the asynchTM. Moreover, the TM is synchronized with the GTM reception, while the SIG is synchronized with the TM reception. The initialization time for this architecture is given in Equation 4.12, where N_{max} is the maximum number of nodes in a sub-network. Furthermore, $N_{dep} \times (C_{GTM} + \epsilon)$ is the longest time to receive the GTM by all master nodes. Moreover, $C_{TM} + C_{asynchTM} + \epsilon$ is required to consider the transmission time of the TM and asynchTM messages to the respective master node. Sending the SIGs for local and global asynchronous messages overlaps with the TRD, Thus the maximum between them must be accounted for.

$$\begin{aligned} init\_time = N_{dep} \times (C_{GTM} + \epsilon) + C_{TM} + C_{asynchTM} + \epsilon \\ + max\{TRD, N_{max} \times (C_{SIG} + C_{asynchSIG})\} \end{aligned} \tag{4.12}$$

The initialization time in the hybrid architecture is determined in Equation 4.13. In the proposed architecture, one TM is used for both synchronous and asynchronous messages compared with the multi-master architecture which uses both the TM and asynchTM. Moreover, one SIG is used to inform the master node about aperiodic messages, instead of using two signals in the multi-master architecture. However, all TM and SIG messages are transmitted through two switches which increase the delay due to the store-and-forward feature of the switches.

$$init\_time = N_{dep} \times (C_{GTM} + \epsilon) + 3 \times C_{TM} \\ +max\{TRD, N_{CL} \times C_{SIG}\} \tag{4.13}$$

N_{CL} is the maximum number of nodes in one cluster and $N_{dep} \times (C_{GTM} + \epsilon)$ is the latest time that GTM is received by the farthest slave node in the network topology. Moreover, $3 \times C_{TM}$ is the TM reception time considering crossing two switches, i.e. from the parent master of a cluster to the slave node. The SIG messages are transmitted in parallel with the TRD, hence the maximum of them should be considered.

In order to compare the architectures with respect to the initialization time, a network with 100 nodes is considered. The depth in the topology (number of levels) is varied, while limiting 30 nodes for each sub-network. The capacity of the network is assumed to be 100Mbps, hence TM and asynchTM messages are assumed $24\mu s$ long. Moreover, SIG messages for both local and global aperiodic traffic are $10\mu s$ long, the GTM transmission time is $8\mu s$, ϵ is $17\mu s$ and TRD is set to $100\mu s$. We observed that the initialization time for the architectures increases with an increasing depth as shown in Figure 4.11. However, in the multi-master architecture the initialization time is much less than what it is in the single-master architecture. In the hybrid architecture the initialization time is even less than what it is in the multi-master architecture.

4.5.2 Data transmission window

Concerning the data transmission window, the goal would be to require the shortest window BW that allows meeting the network requirements of a given application. Thus, we compute $rbf_i(t)$ assuming a given constant EC in Equation 4.1, and for all $t \in CP_{rbf}$ the BW is derived such that $rbf_i(t) = sbf(t)$. The minimum BW among those computed for all check points is the window required for that message to meet its deadline. Then, BW is computed for all messages and the maximum value is the minimum window that makes all messages schedulable. To show the difference in the data transmission window between the architectures, we apply the analysis of each architecture to two different scenarios. In the first one, the architecture is the same as in Figure 4.3 with 20 nodes distributed in the network. In the second scenario, we use the same topology but with 70 nodes distributed in the network. In both cases, we generate 100 instances for two different sets, one with 100 and another with 500 messages, i.e., we generate 100 sets randomly for two different

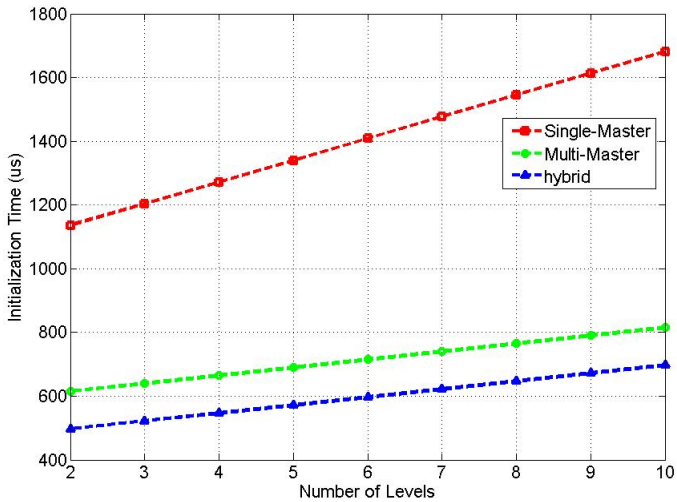

Figure 4.11: Initialization time (constant nodes).

cases, a set of 100 messages and a set of 500 messages. The average window computed for the 100 instances of each set is presented for the two network scenarios in Figure 4.12. Moreover, to include the effect of initialization time in the examples, we sum it with the synchronous and asynchronous windows in Figure 4.12. The vertical lines in Figure 4.12 shows the deviation of minimum and maximum calculated synchronous and asynchronous windows in all 100 instances. The deviation states that the average values of different architectures are representative for the evaluation.

The results show that, for a low number of messages in both the 20-node and the 70-node examples, the single-master case needs shorter windows for synchronous and asynchronous traffic compared with the multi-master architecture. The reason is that each window in the multi-master architecture is divided into different types, i.e., global and local which cause a significant increase in terms of idle times that are assumed in each sub-window. However, the initialization time for the single-master architecture increases when the number of nodes increases, in agreement with Figure 4.11. Therefore, the time needed within an EC to transmit the traffic increases in case of the 70-node example, even though the data transmission window is smaller compared

to that of the multi-master architecture. Nevertheless, by increasing the number of messages to 500, in both the 20-node and the 70-node examples, the synchronous window in the multi-master architecture is much smaller compared to that of the single-master architecture. The reason is that in the single-master the switching delays of local traffic are accumulated to global traffic which results in an allocation of a larger window. However, the asynchronous window in the multi-master is larger due to having more idle time, i.e., due to a higher number of partitioned windows for the clusters. Finally, considering initialization time the multi-master architecture requires a smaller portion of the EC to transmit the traffic, in particular for a network with a higher number of nodes. Considering the hybrid architecture, the results show that the window needed for transmitting the data is much smaller compared both to the single-master and to the multi-master architectures. This is highlighted in particular for networks with a high number of nodes and messages. Moreover, taking the initialization time into account, the time needed for data transmission plus the initialization time in the hybrid architecture for large networks is much lower compared to that of the other two architectures.

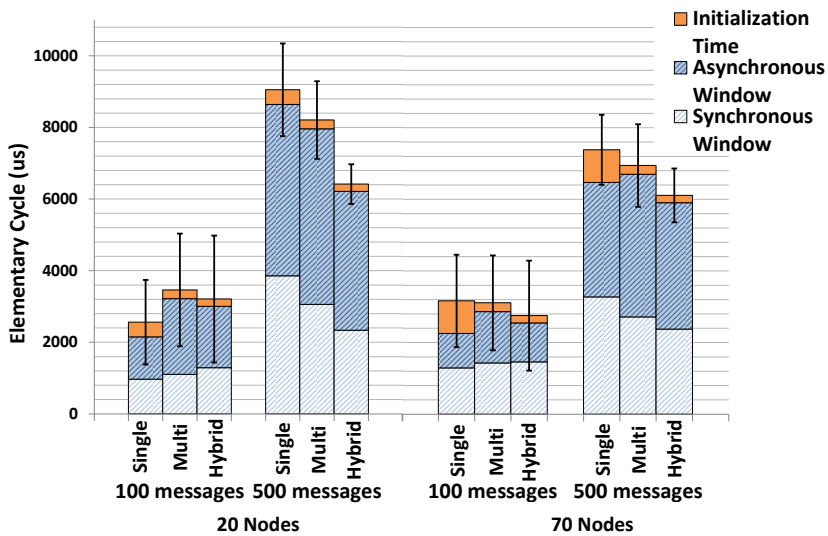

Figure 4.12: Data transmission window requirement.

4.5.3 Scheduler overhead

The master nodes in the architectures schedule the ready messages in parallel with the transmission of data messages. The time needed to schedule the messages depends on the processing capacity of the master node and on properties of the message set. However, the scheduling process should be finished before the start of the next EC to avoid missing cycles. The time which is allocated for the master node to schedule the ready messages is $(EC - InitializationTime)$. Therefore, according to Figure 4.11 and considering a constant EC, the time available for the master node to schedule the messages is decreased when increasing the number of levels in the network topology. This scheduling time is tighter for the single-master architecture. Moreover, in the single-master architecture, one master node is responsible for scheduling all the traffic in the network, while in the multi-master architecture this duty is divided among several master nodes for local messages. Therefore, this situation is more demanding in the single-master architecture with less time available for scheduling. In addition, in the hybrid architecture, the number of master nodes used to coordinate the traffic is noticeably lower than in the multi-master architecture while still having the benefits of that architecture. For instance, the network in Figure 4.3 requires two master nodes in the hybrid architecture and five in the multi-master architecture, which makes the former more cost effective as well.

4.6 Conclusions

This chapter presented two proposals to extend the single-switch FTT-SE protocol to enable multi-hop communication. For this purpose, two architectures are presented, being the multi-master and hybrid architectures. An improved response time analysis for the architectures is presented. In addition, the performance of the architectures with respect to data transmission window, initialization time and scheduler overhead is evaluated. In general, the results showed that the multi-master architecture has better resource utilization for large networks compared with a single-master architecture. Moreover, the hybrid architecture provides the benefits of previous two architectures, i.e, the single-master and multi-master architectures. Finally, the evaluation showed that the overall performance is improved in the hybrid architecture.

Chapter 5

Multi-Hop HaRTES Architecture

This chapter presents two solutions to extend the single-switch HaRTES architecture in order to provide multi-hop communication. The first solution uses the HaRTES switch ability in buffering the incoming messages, while the second solution provides a method to forward the messages with as little as possible buffering in their route. Response time analyses for both solutions are also proposed in this chapter. Finally, a set of comparisons is presented between the solutions to show their respective advantages.

5.1 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 5.1 illustrates 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 (for instance, sensors, controllers and 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*. The switch on the top of the tree topology is called the *root switch*.

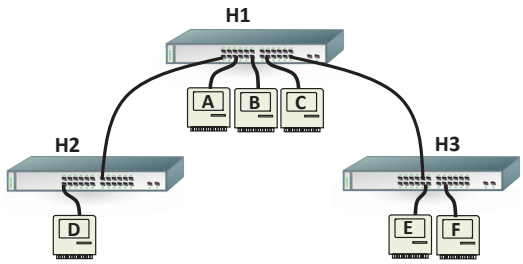

Figure 5.1: An example of the multi-hop HaRTES network.

5.2 Distributed Global Scheduling (DGS)

This section presents a scheduling method in the multi-hop HaRTES architecture, which preserves the basic properties of the HaRTES architecture. This method is named Distributed Global Scheduling (DGS), where all switches in the route of a message cooperatively schedule the message.

5.2.1 Scheduling algorithm

In the DGS method, the scheduling of global traffic is carried out in a distributed fashion by all involved switches. Basically, each switch schedules its hop without distinction of global or local messages. Firstly, the switch to which the source node is directly connected to, schedules the message for transmission and stores it in its own memory, thus behaving as a consumer of the message. Then, in a posterior EC, the next switch in the route schedules the message for transmission, which is sent by the first switch and stored in the local memory of the second switch. The step-wise scheduling continues until the last switch, for which the destination node is connected to. The last switch schedules the message to be received and transmitted to the destination node, as the consumer of the message is attached to that switch. Thus, the message is not buffered in the last switch and it is immediately forwarded to the destination node in one EC.

In order to keep the time-triggered model for synchronous global traffic, the messages are activated periodically in the switches considering a *phase* defined for each message. The phase for a message is defined differently in

each switch in the route of the message, and it determines a time delay between the activation time in the switch where the message is being scheduled and the activation time in the source node. The phase is specified in number of ECs and is essential to guarantee that, in each hop, messages are always forwarded after being received from the previous switch.

To illustrate the operation of the message forwarding process, consider the network depicted in Figure 5.1 and the scheduling trace in Figure 5.2. Message m_1 shall be transmitted from node D, connected to switch H2, to node E, connected to switch H3. Firstly, H2 schedules m_1 and stores it in its own local memory (EC_k). In the following EC the message is scheduled by switch H1, being transferred from the internal memory of H2 to the internal memory of H1 (EC_{k+1}). Finally, the message is scheduled by H3, being transmitted from the internal memory of H1 to node E (EC_{k+2}). In the first two ECs there is no impact of the switching delay on the message response time since the message is buffered in the switch, while in the last EC the switching delay needs to be accounted for, as the message is forwarded to the destination node without being buffered.

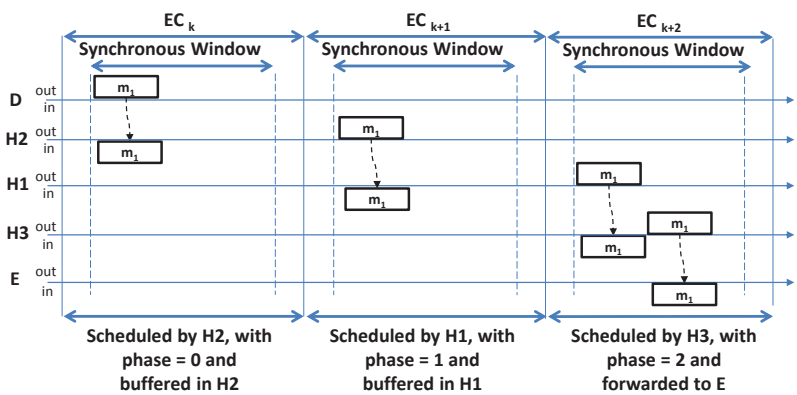

Figure 5.2: The operation of the DGS scheduling method.

Note that this is a very simple case. In reality messages may be delayed several ECs in each switch if several higher priority messages, exceeding the capacity of the EC, are waiting to be transmitted in the link that connects the switches. Thus, the phase for a message, to be taken into account in each hop, is the worst-case response time of the message from the source node to that hop, i.e, the maximum delay accumulated in the ancestor links.

The definition of consistent phases in the transmission of synchronous messages across multiple switches requires global synchronization. The HaRTES architecture guarantees a contention-free transmission of the TM, which is broadcasted to all the nodes connected to a switch, including other switches down the tree topology. Therefore, the TM can be used as a precise time mark. In this proposal, all switches synchronize with their parent switch whenever receiving a TM, hence providing synchronized ECs. The exception is the root switch, which behaves as a time master. Other clock synchronization techniques based on the IEEE 1588 standard can also be used, as presented in similar cases by Ashjaei *et al.* [154]. However, the clock synchronization is excluded from this work.

The asynchronous traffic is forwarded by the switches, within the asynchronous windows through a hierarchy of servers. During the synchronous windows, or when the capacity of associated servers is exhausted, such traffic is suspended and queued.

5.2.2 Response time analysis

In this section, we present the response time analysis for the single-switch HaRTES architecture. Then, we extend the presented analysis for the DGS method.

Single-switch response time analysis

The scheduling policy in the HaRTES switch is based on resource reservation in each link which presents a resemblance with the periodic model in hierarchical scheduling [138]. Therefore, the associated analysis based on a *request bound function (rbf)* and a *supply bound function (sbf)* is a suitable method for evaluating the response time of the messages in this network. The $rbf(t)$ represents the maximum load generated by a message and the $sbf(t)$ is the minimum effective communication capacity that a link in the network provides.

As the period and deadline for messages are stated in number of ECs, the response time of m_i is also computed in number of ECs. The computation of response time is done in Equation 5.1, where $\theta_i^{l_s,l_d} = min(t > 0) : sbf_i^{l_s,l_d}(t) \geq rbf_i^{l_s,l_d}(t)$. The response time is calculated in the route of the message that contains the source node link (l_s) and the destination link (l_d).

$$RT_i^{l_s,l_d} = \left\lceil \frac{\theta_i^{l_s,l_d}}{EC} \right\rceil \tag{5.1}$$

$rbf_i^{l_s,l_d}(t)$, as shown in Equation 5.2, is computed by summing the transmission time of m_i, the interference from other messages and the switching delay of the interfering messages as well as the switching delay of m_i itself.

$$rbf_i^{l_s,l_d}(t) = C_i + W_i^{l_s,l_d}(t) + Is_i^{l_s,l_d}(t) \tag{5.2}$$

The interference from other messages is caused by the messages with the same or higher priority than that of m_i which share link l_s or l_d with m_i and it is denoted by $W_i^{l_s,l_d}(t)$. This type of interference is called *Shared Link Delay*. In order to calculate the Shared Link Delay, all higher or same priority messages that share link l_s or l_d with m_i should be accounted for. The computation is shown in Equation 5.3.

$$W_i^{l_s,l_d}(t) = \sum_{\substack{\forall j \in [1,N] \\ \wedge\ m_j \in hep(m_i) \\ \wedge (l_s \in \mathcal{L}_j \vee l_d \in \mathcal{L}_j)}} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{5.3}$$

The switching delay of interfering messages is combined with the switching delay of message m_i, in order to bound its impact in the transmission window. The switching delay interference is denoted by $Is_i^{l_s,l_d}(t)$. To determine the switching delay interference $Is_i^{l_s,l_d}(t)$, we define a multi-set $G_i^{l_s,l_d}(t)$ which contains the switching delays of all messages that contribute to the Shared Link Delay and are activated within interval $[0, t)$ as well as the switching delay of m_i. Such a method was introduced by Behnam *et al.* [152] to improve the response time analysis of a synchronization protocol in a hierarchical scheduling framework. A message may be activated several times within an interval. Therefore, all activation instants of a message need to be added to the multi-set. The multi-set is formulated in Equation 5.4, where y shows the number of instances that message m_j is activated, hence y number of switching delays for that message should be added to the multi-set.

$$\begin{aligned} G_i^{l_s,l_d}(t) = \ & \{ \bigcup_{y=1..\lceil \frac{t}{T_j} \rceil} (SWD_j + \epsilon) : m_j \in hep(m_i) \\ & \wedge (l_s \in \mathcal{L}_j \vee l_d \in \mathcal{L}_j) \} \cup (SWD_i + \epsilon) \end{aligned} \tag{5.4}$$

Tracking time t, in the analysis, one element per EC will be taken from $G_i^{l_s,l_d}(t)$ and to consider the worst-case scenario, the largest element will be selected. Therefore, we define a sequence $\left(G_i^{l_s,l_d}(t)\right)^{sort}$ that includes the switching delay values from $G_i^{l_s,l_d}(t)$ in a descending order. The computation

of the switching delay interference is shown in Equation 5.5, where $z(t)$ denotes the number of ECs that have passed, hence the number of elements from $\left(G_i^{l_s,l_d}(t)\right)^{sort}$.

$$Is_i^{l_s,l_d}(t) = \sum_{k=1}^{z(t)} \left(G_i^{l_s,l_d}(t)\right)^{sort}[k] \tag{5.5}$$

$$z(t) = \left\lceil \frac{t}{EC} \right\rceil \tag{5.6}$$

$sbf_i^{l_s,l_d}(t)$ is the communication capacity that links l_s and l_d provide in the time interval $[0, t)$. Note that each EC comprises a window dedicated to each type of traffic and a different length can be set for each link (i.e., LW^{l_s} for the source link and LW^{l_d} for the destination link). The minimum guaranteed window size, which may actually be used to carry messages, is smaller than the nominal value due to the need of preventing window overruns, being given by $LW^{l_s} - Id_i^{l_s}$ for the source link, as an example. The idle time is upper bounded by the maximum packet size of m_i and all messages with a priority higher than that of m_i sharing link l_s or l_d with m_i. Therefore, the supply bound function is computed in Equation 5.7, where the idle time for link l is computed in Equation 5.8. Note that the window length provided in the source link may be different from the destination link. In order to consider the worst-case scenario we take the minimum between them in Equation 5.7.

$$sbf_i^{l_s,l_d}(t) = \frac{min(LW^{l_s} - Id_i^{l_s}, LW^{l_d} - Id_i^{l_d})}{EC} \times t \tag{5.7}$$

$$Id_i^l = \max_{\forall m_p \in hep(m_i) \wedge\ l \in \mathcal{L}_p} \{PK_p\} \tag{5.8}$$

It should be noted that the $sbf_i^{l_s,l_d}(t)$ presented in Equation 5.7 is the approximation of the supply bound function. However, as we calculate the response time in number of ECs, the approximation does not introduce additional pessimism. Figure 5.3 shows the exact $sbf_i^{l_s,l_d}(t)$, the approximation of that and the $rbf_i^{l_s,l_d}(t)$ for m_i. The $rbf_i^{l_s,l_d}(t)$ meets both exact and approximate $sbf_i^{l_s,l_d}(t)$ in the third EC. Thus, the response time using both $sbf_i^{l_s,l_d}(t)$ and the approximation is 3 EC. Therefore, we can use the approximation of the supply bound function in this particular network protocol to calculate the exact response time.

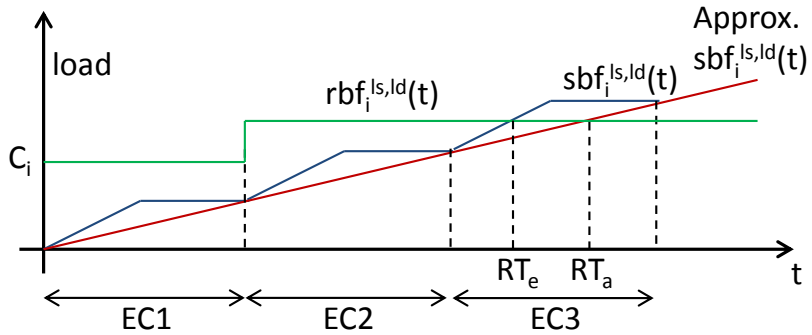

Figure 5.3: The approximation of the supply bound function.

Determining $\theta_i^{l_s,l_d}$ in Equation 5.1 requires checking the inequality at all instants where $rbf_i^{l_s,l_d}(t)$ changes due to the interference of other messages. Such a set of checkpoints is given by Equation 5.9.

$$CP_{rbf_i^{l_s,l_d}} = \{\bigcup cp_{m_a} : m_a \in hep(m_i) \wedge (l_s \in \mathcal{L}_a \vee l_d \in \mathcal{L}_a)\} \cup \{T_i, C_i\} \tag{5.9}$$

$$where, cp_{m_a} = T_a, 2T_a, ..., \alpha T_a, \alpha = \left\lfloor \frac{T_i}{T_a} \right\rfloor$$

Multi-hop response time analysis

In the DGS method, the global messages are buffered in each switch before being scheduled by the next switch. Therefore, the response time of the global messages is the sum of response times in each switch, which is calculated separately. Moreover, local messages are transmitted through one switch within the same window as global messages (i.e., synchronous window). Thus, the global and local messages that share links may interfere with each other.

Global messages are buffered in each switch, except in the last switch where they are forwarded to the destination node in the same EC, i.e., the last switch acts as a single-switch case. Therefore, response time analysis in the last switch is computed similarly to the analysis for the single-switch architecture. However, in the other switches, the messages are scheduled for the input links, only. Thus, the response time analysis should be performed for each one

of the input links of the switches. The response time for a global message m_i is calculated in Equation 5.10, where l is the set of links in the route of the message except the last two links (i.e., the link to the last switch denoted by l_x and the link to the destination node denoted by l_d). The response time in the last switch $RT_i^{l_x,l_d}$ is computed similarly to the single-switch case.

$$RT_i = \sum_{\forall l \in \mathcal{L}_i \wedge\ l \neq l_x \neq l_d} RT_i^l + RT_i^{l_x,l_d} \tag{5.10}$$

The response time in link l is computed in number of ECs as shown in Equation 5.11, where $\Theta_i^l = min(t > 0) : sbf_i^l(t) \geq rbf_i^l(t)$.

$$RT_i^l = \left\lceil \frac{\Theta_i^l}{EC} \right\rceil \tag{5.11}$$

$rbf_i^l(t)$ is calculated similarly to Equation 5.2, except considering l instead of two links (l_s, l_d). Moreover, thanks to the buffering feature of the switch, the switching delay is not applicable as we are not forwarding the messages in the same EC, i.e., $Is_i^l(t) = 0$. Therefore, $rbf_i^l(t)$ is computed in Equation 5.12.

$$rbf_i^l(t) = C_i + W_i^l(t) \tag{5.12}$$

The Shared Link Delay ($W_i^l(t)$) is the interference from message with the same or higher priority than that of m_i, which share link l with m_i. This delay is computed in Equation 5.13.

$$W_i^l(t) = \sum_{\substack{\forall j \in [1,N] \\ \wedge\ m_j \in hep(m_i) \wedge \\ l \in \mathcal{L}_j}} \left\lceil \frac{t}{T_j} \right\rceil C_j \tag{5.13}$$

$sbf_i^l(t)$ is also computed similarly to Equation 5.7, except there is one link l available for message transmission. Therefore, the supply bound function is calculated in Equation 5.14, where Id_i^l is computed in Equation 5.8.

$$sbf_i^l(t) = \frac{LW^l - Id_i^l}{EC} \times t \tag{5.14}$$

In order to determine the checkpoints in which $rbf_i^l(t)$ changes due to the interference of other messages, we verify the checkpoints similarly to Equation 5.9, except considering one link l instead of l_s and l_d. Note that, the response time analysis for local messages is performed similarly to the single-switch case.

5.2.3 Validation of DGS

This section presents two different evaluations. Firstly, the validity of the presented response time analysis is checked by comparing the results observed from simulation with the ones calculated by the analysis, using a high loaded example. Then, the applicability of the DGS method in an industrial setting is validated by applying this method in an automotive case study.

5.2.4 Analysis evaluation

In this evaluation, we assess the level of pessimism embodied in the analysis by loading up the links in a network example. In order to simulate the example we have used a simulation tool called SEtSim [153]. In order to evaluate the analysis, we considered a network comprising 4 switches along with 10 nodes, as depicted in Figure 5.4.

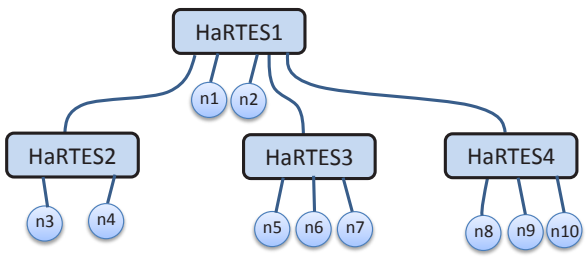

Figure 5.4: A network example for evaluation of DGS.

We defined 28 messages that include both local and global traffic. The parameters of the defined messages are shown in Table 5.1. From the defined messages we tagged two of them, one local and one global, and we generated an unfavorable situation for them. The tagged message m_1, as a global message, and m_2, as a local message are selected. In order to increase the interference for the tagged messages, we assigned the lowest priority to them. Moreover, for the global tagged message, we selected the longest route in the network to increase the total interference that it is subject to, which accumulates over each link that the message crosses. The other messages (m_3 to m_{28}) are defined to interfere with the two tagged messages such that they generate a long delay for them. The periods of the interfering messages are selected

within $[4, 12]EC$ and the priorities of them are selected according to the RM algorithm. Moreover, the transmission time of the interfering messages are chosen within $[100, 123]\mu s$, where $123\mu s$ equals to 1542 bytes as the maximum Ethernet frame size. Note that a higher value for P_i equals to a lower priority (i.e., 1 presents the highest priority). In this example, the network capacity is set to $100Mbps$ and the switch latency is $\epsilon = 3\mu s$. Furthermore, the EC is considered to be $1ms$. We also assigned $600\mu s$ to the synchronous window within the EC. We simulated the above described example for a hyper period (least common multiple of the periods), which is 27720 ECs. Figure 5.5 illustrates the maximum response time of the messages measured from the simulation and the ones computed using the proposed analysis. In this figure, the x-axis represents a message id and the y-axis shows the message response time in number of ECs.

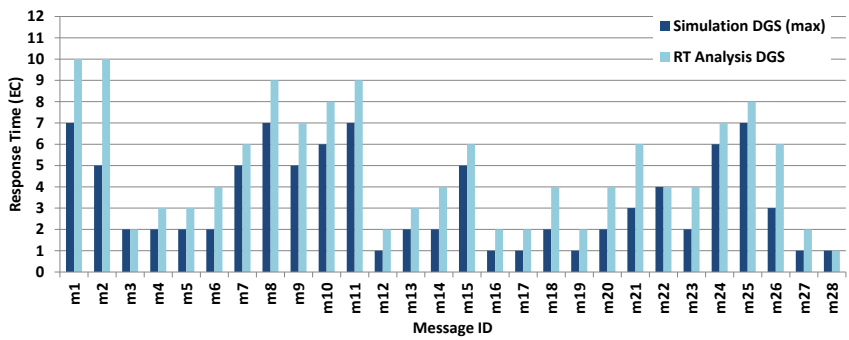

Figure 5.5: Response time of messages using the DGS method.

ID	T=D (EC)	P	C(μs)	S	Ds	G/L
m_1	12	10	123	n3	n8	G
m_2	12	10	123	n5	n6	L
m_3	5	1	123	n3	n1	G
m_4	6	2	123	n3	n2	G
m_5	7	3	100	n4	n1	G
m_6	8	4	123	n4	n2	G
m_7	9	5	100	n3	n8	G
m_8	9	6	123	n4	n9	G
m_9	10	7	123	n5	n8	G
m_{10}	11	8	123	n6	n9	G
m_{11}	11	9	100	n7	n10	G
m_{12}	12	10	110	n9	n8	L
m_{13}	4	1	123	n1	n9	G
m_{14}	6	2	123	n1	n10	G
m_{15}	9	4	100	n4	n10	G
m_{16}	5	1	100	n7	n6	L
m_{17}	6	2	123	n7	n6	L
m_{18}	7	3	123	n5	n6	L
m_{19}	7	6	123	n5	n7	L
m_{20}	10	7	123	n5	n6	L
m_{21}	10	8	110	n5	n6	L
m_{22}	12	10	123	n7	n6	L
m_{23}	4	1	123	n1	n6	G
m_{24}	6	2	123	n8	n6	G
m_{25}	7	3	110	n10	n6	G
m_{26}	10	8	123	n5	n6	L
m_{27}	4	1	123	n7	n6	L
m_{28}	6	2	123	n5	n7	L

Table 5.1: The parameters of the tagged messages in DGS example.

As we can see from the results, the analysis introduces a pessimism for both local and global messages. The response time measured in the simulation for m_1 (as the global tagged message) is $7EC$, while the analysis calculates $10EC$. Moreover, the local tagged message (m_2) has $10EC$ computed response time, while it is measured $5EC$ in the simulation.

We believe that the pessimism in the analysis stems from two different sources. The first source of pessimism is the phase of messages, described in the DGS method, which is not considered in the response time analysis. As we observed in the simulation, neglecting the phase in the analysis is the main reason of the pessimism. However, adding that to the analysis is not straightforward and it is one of the directions for future work. In order to describe the pessimism cause, assume a simple network example depicted in Figure 5.6. Moreover, consider two messages, m_1 transmitted from n1 to n2 and m_2 transmitted from n3 to n2. Let us assume that m_2 is the lower priority message and that each message needs one EC for transmission. The defined messages share the n2 downlink. Therefore, when computing the response time for m_2, the interference from m_1 is added.

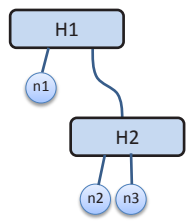

Figure 5.6: A network example to show the pessimism in DGS analysis.

The scheduling window is depicted in Figure 5.7. m_1 is transmitted to switch H1 in EC1, then it is forwarded to n2 in EC2. m_2 is transmitted in the first EC, and m_1 does not interfere in that EC. Therefore, m_1 does not delay m_2 as it is transmitted with a 1EC phase. Considering this example, the response time for m_2 is 1EC, while by adding the interference of m_1 in the analysis, it is computed to be 2EC.

The second source of the pessimism is found in the calculation of the switching delays for the interfering messages. For the global messages we consider the switching delay in the last switch of the route of the message, only. However, it can produce a pessimism when we define several interfering messages sharing links with the message under analysis in the last switch. The reason is that the scheduler takes the maximum switching delay of the messages scheduled for each EC, while in the analysis, as we do not have such an information, we consider the maximum switching delay among all messages that are activated during an interval of time.

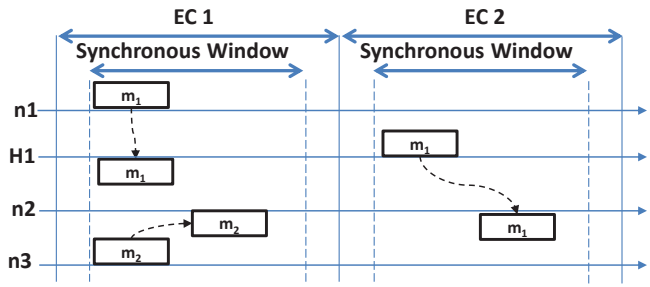

Figure 5.7: Scheduling window for pessimism example in DGS.

5.2.5 Automotive case study

Currently the interest of using high bandwidth network technologies in automotive industries is growing rapidly. This demand is inherent in the use of innovative applications as well as entertainment devices in a vehicle. Lim *et al.* [155, 156] investigated the performance of in-car switched Ethernet using real traffic sets in a vehicle. The former work studies the use of traffic without prioritization in a switched Ethernet network, whereas the latter work investigates the average end-to-end delay of the same traffic set with prioritization and scheduling using a weighted fair queuing. In this case study, we use the same set of traffic as in [155] and we define a network using HaRTES switches on which we apply the DGS method.

The network comprises 12 nodes, including the Head Unit, cameras, a multimedia node and control nodes. The network topology for this case study is depicted in Figure 5.8, where the nodes are connected according to the geometrical arrangement of nodes in a vehicle (HaRTES2 is in the front and HaRTES3 is in the rear of the vehicle).

There are three different types of traffic including control data, camera streaming, video and audio streaming. The parameters of the traffic are given in Table 5.2. In this example, all traffic types are synchronous. We set the EC length to $2ms$, where we, within the EC, assigned $1700\mu s$ to the synchronous window. The maximum end-to-end delay of the messages as they are measured from the simulation and their corresponding deadlines in number of ECs are shown in Table 5.3. The more critical messages in this set are the control messages (m_1 to m_4) with very short and firm deadlines. The other messages have larger deadlines as they are audio and video streams. As it can be seen

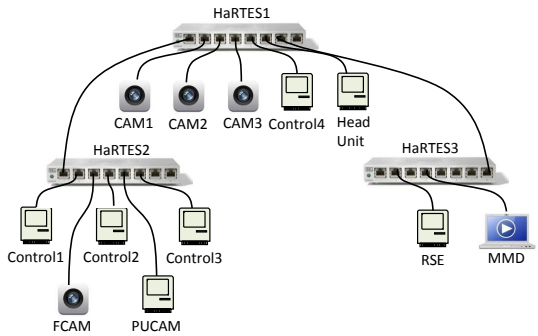

Figure 5.8: Network topology for automotive case study in DGS.

ID	Type	Packet Size	T (ms)	D (ms)	P	S	Ds
m_1 to m_4	Control	20	10	10	1	Control 1 to 4	Head Unit
m_5 to m_7	Camera	6288	4	45	2	CAM 1 to 3	Head Unit
m_8	Fcam	6288	4	45	2	FCAM	PU CAM
m_9	Audio	1472	10	150	3	MMD	RSE
m_{10}	Video	14720	10	150	3	MMD	RSE

Table 5.2: Traffic parameters for automotive case study.

from the results, all deadlines are met by observing the response time of the messages using the simulation.

ID	D(EC)	RT(EC)	ID	D(EC)	RT(EC)
m_1	5	2	m_6	22	2
m_2	5	2	m_7	22	2
m_3	5	2	m_8	22	1
m_4	5	1	m_9	75	1
m_5	22	2	m_{10}	75	1

Table 5.3: Response time of the automotive case study in DGS.

5.3 Reduced Buffering Scheme (RBS)

This section presents an alternative method, namely the Reduced Buffering Scheme (RBS), to schedule and forward messages in the multi-hop HaRTES architecture. However, this solution requires a modification in the HaRTES switch. Therefore, the switch modification is presented beforehand. Moreover, a response time analysis along with the validation of the method are presented for the RBS method.

5.3.1 HaRTES switch modification

The HaRTES switch is implemented using 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 implementation alternatives that present different trade-offs between performance and resource utilization [157].

Several options have been considered and the one 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 5.9 shows the internal architecture of the output port management. To make the figure more readable, it is represented using 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, stores messages 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.

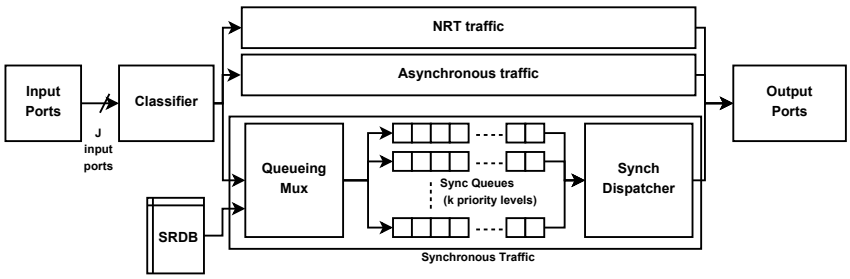

Figure 5.9: Modified HaRTES switch architecture.

The amount of resources depends on the number of messages and priorities. In particular, each implemented priority level requires a static allocation of a portion of the memory sufficiently large 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.

5.3.2 Scheduling algorithm

In the RBS method all links are partitioned between two windows, the synchronous and asynchronous windows, the same way as in the DGS method. The allocated window size in each link is differentiated and it is selected based on the actual local and global load crossing that link. Different sizes for the transmission windows increases the efficiency of using the bandwidth in the links. Figure 5.10 illustrates the EC partitioning in the RBS method. A particular window (Guard Win) is reserved in the beginning of each EC for different purposes in the RBS method, which is described in this section.

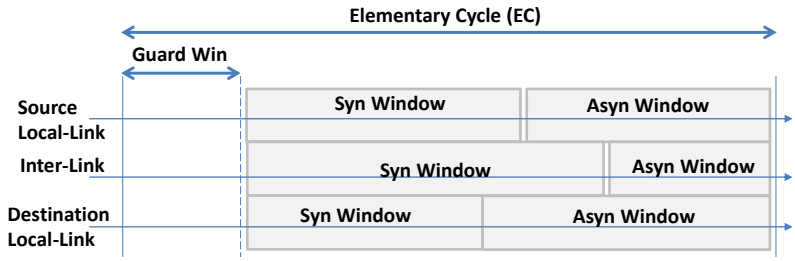

Figure 5.10: The EC partitioning in the RBS method.

The switch to which the source node of a synchronous message is connected to, schedules the message similarly to the single switch case and informs the source node using the TM. The TM is transmitted at the beginning of the EC within the Guard Window (Figure 5.10).

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

Figure 5.11 depicts the same scenario for m_1 presented in Figure 5.2, yet using the RBS method. In EC_k the message is scheduled by switch H2 to be sent from its source node (node D). Then, it is received by switch H2 and inserted to the priority queue in the output link. As there is still enough time in the synchronous window in EC_k, switch H2 forwards m_1 to switch H1. Similarly to switch H2, switch H1 forwards m_1 to switch H3 as there is still enough time in the synchronous window. However, the transmission of m_1 is suspended in switch H3 due to the lack of remaining time in the synchronous window. In EC_{k+1} the transmission is resumed and m_1 is received by the destination node (node E).

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

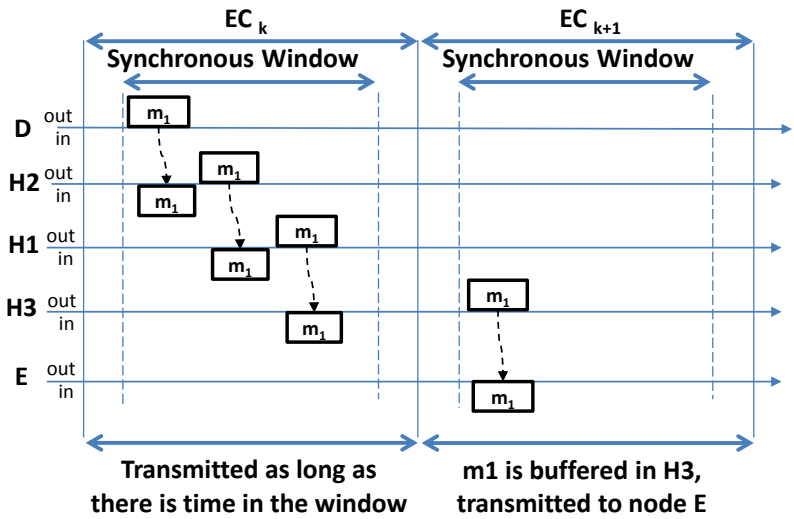

Figure 5.11: The operation of the RBS method.

The time synchronization among the HaRTES switches is required in order to increase the efficiency of the RBS method. In this context, a clock synchronization according to the IEEE 1588 can be used, which was proposed by Ashjaei *et al.* [154] for a similar case. The required signaling is carried out within the Guard Window. However, a detailed study of the clock synchronization for the multi-hop HaRTES architecture is out of the scope of this dissertation.

5.3.3 Response time analysis

In the response time analysis, we capture the RBS behavior by calculating the response time link-by-link from the source node and check whether the message is buffered in any switch connected to that link.

Suppose that, we are calculating the response time of m_1 that is transmitted from node D to node F in the network depicted in Figure 5.1. First, we compute the response time for the link from node D to switch H2, i.e., the source local-link. We continue to compute the response time for two links from node D to switch H1. Then, we compare the two computed response times (in number of ECs), i.e., the one for the source link and the one from node D to switch H1. If they are equal, we continue to calculate the response time for three links from

node D to switch H3. However, if they are not equal, we save the response time of the source link to the total response time and we start computing the response time from the link between switch H2 to switch H1. We continue the same way until the last link, i.e., the destination (node F) link. This way we capture the behavior of the RBS method that has a combination of buffering and forwarding instances for a message through switches. Algorithm 3 illustrates such a calculation.

Algorithm 3 Response time calculation for m_i in RBS.

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

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

When the algorithm computes the response time in one link (i.e., $l_a = l_b$), the previous response time is not available to compare with the latest response time. In this case, the loop continues for the next link in the route of m_i. This checking is carried out in line 6 and continues in line 9. In contrast, if

the response time computation is for several links from link l_a to link l_b, and if the latest response time is not equal to the response time calculated until the previous link (line 6), the message m_i is buffered in the previous switch. Therefore, the algorithm stops calculating and adds up the calculated response time until previous link $l_{(b-1)}$ to the total response time RT_i (line 7). This means that, we calculate the response time for m_i until link $l_{(b-1)}$, where the message is buffered. Then, the algorithm commences to compute the response time from link l_b. Thus, line 8 sets the starting link l_a to the link where 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. For this calculation, 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 3 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 Equation 5.15 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).

$$\alpha_{i,a,b} = \frac{\min\limits_{l=a..b}(LW_l - Id_{i,l})}{EC} \tag{5.15}$$

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 Equation 5.16.

$$Id_{i,l} = \max_{\substack{\forall r \in [1,N] \\ \wedge\ m_r \in hep(m_i) \\ \wedge\ l \in \mathcal{L}_r}} (PK_r, PK_i) \tag{5.16}$$

The response time of m_i, shown in line 4 in Algorithm 3, is evaluated iteratively in Equation 5.17 by considering the transmission time of the message

itself and the interference from other messages. The interference of other messages is categorized in three types: (i) the interference from messages, that share links between link l_a and l_b with m_i, with higher or the same priority than that of m_i, which is specified by $I_{i,a,b}$, (ii) the blocking from messages with priority lower than that of m_i sharing 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.

$$rt_{i,a,b}^{(x)} = \frac{C_i}{\alpha_{i,a,b}} + I_{i,a,b} + B_{i,a,b} + SD_{i,a,b} \tag{5.17}$$

The iteration can start from $rt^{(0)} = \frac{C_i}{\alpha_{i,a,b}}$, and the response time of m_i between link l_a and link l_b is calculated in Equation 5.18.

$$rt_{i,a,b} = rt_{i,a,b}^{(x)} \; when \; rt_{i,a,b}^{(x)} = rt_{i,a,b}^{(x-1)} \tag{5.18}$$

The first interference term in Equation 5.17 is caused by the higher and the same priority synchronous messages 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 Equation 5.19. Note that, the interference should be inflated by the same inflation factor.

$$I_{i,a,b} = \sum_{\substack{\forall j \in [1,N], j \neq i \\ \wedge \; m_j \in hep(m_i) \\ \wedge \; \mathcal{L}_j \cap \mathcal{L}_{i,a,b} \neq \emptyset}} \left\lceil \frac{rt_{i,a,b}^{(x-1)}}{T_j} \right\rceil \frac{C_j}{\alpha_{i,a,b}} \tag{5.19}$$

According to the RBS method, a message received by a HaRTES switch is inserted to the output priority queue. If there is enough time in the EC, the message is transmitted to the next switch. However, it may happen that, concurrently with the insertion, a lower priority message is transmitted through the same priority queue. Therefore, the arrival message is blocked with the lower priority message. Note that, one particular lower priority message can block m_i only once in the route. Therefore, the same blocking messages in the next links are excluded from the calculation. Also note that, the blocking is at most one packet as the other packets are preempted by m_i. The blocking for m_i is calculated in Equation 5.20.

$$B_{i,a,b} = \sum_{t=a+1..b, a\neq b} \max_{\substack{\forall p\in[1,N] \\ \wedge\ m_p\in lp(m_i) \\ \wedge\ l_t\in\mathcal{L}_p \\ \wedge\ \forall y, a+1\leq y<t, l_y\notin\mathcal{L}_p}} \left(\frac{PK_p}{\alpha_{i,a,b}}\right) \tag{5.20}$$

In the RBS method, synchronous messages are transmitted by the source node when indicated in the TM. Therefore, in the source 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 Equation 5.20 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 Equation 5.17 is the switching delay of the message. As it is described in the system model, the switching delay is the delay of buffering an arrival message before transmitting. This delay is different than the blocking and the interference. The switching delay occurs for a message crossing a switch even without being blocked or delayed by another message. However, the switching delay of other messages that share links with the message under analysis m_i do not affect the switching delay of m_i. In order to show the effect of switching delay of a message on other messages, we consider three different cases that cover all possible scenarios for m_1 and m_2 transmitted through one switch as follows:

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

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

Case 3: m_1 and m_2 share both input and output links. In this scenario the first message is stored in the switch and transmitted to the output link. During the transmission of the first message, the second message is being stored and wait for the transmission. Figure 5.12 shows the transmission of these messages. The switching delay of m_2 is not equal to its transmission time, and in fact it is equal to the transmission time of m_1, assuming $C_1 > C_2$. This is due

to the storing time of m_2 which is carried out concurrently with the transmission of m_1. Thus, when calculating the switching delay for m_2, the switching delay of m_1 should be considered. This scenario occurs when both messages are transmitted in series from the input to the output of a switch.

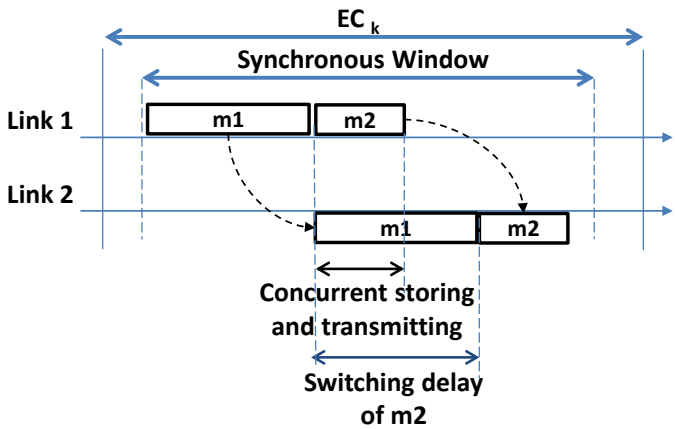

Figure 5.12: Switching delay example in the RBS method.

In general, we can conclude that, the switching delay for m_i is the maximum transmission time between the synchronous messages that share both input and output links with m_i, and the message itself. Note that, all messages including high and low priority messages are taken into account. The reason is that, there might be some lower priority messages ahead of m_i due to the blocking situation in the previous switches. The switching delay for m_i transmitted between link l_a and link l_b is calculated in Equation 5.21.

$$SD_{i,a,b} = \sum_{t=a+1..b, a\neq b} \max_{\substack{\forall q\in[1,N] \\ \wedge\ l_t\in\mathcal{L}_q \\ \wedge\ l_{(t-1)}\in\mathcal{L}_q}} \left(\frac{SWD_i, SWD_q}{\alpha_{i,a,b}} \right) \tag{5.21}$$

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 $l_{(a+1)}$.

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 link, in contrast with the synchronous messages, i.e., $B_{i,1,1} \neq 0$. However, in links other than the source link, when computing the response time for one link, the blocking does not exist as the message was buffered and the concurrent transmission with a possible lower priority message did not occur. Thus, $B_{i,a,b} = 0$ when $l_a = l_b \neq l_1$. In addition, the asynchronous messages are pended during the synchronous window. Thus, they cannot interfere with the synchronous messages. Therefore, when calculating the response time the interfering messages and blocking messages are the asynchronous instead of synchronous. Also, the inflation factor is calculated considering the asynchronous window for LW. Algorithm 3 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 with equations 5.17 and 5.18.

The complexity of Algorithm 3 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 3, is pseudo-polynomial. Therefore, the complexity of the algorithm remains pseudo-polynomial.

5.3.4 Experimental evaluation

In order to validate the RBS method, the scheduling and forwarding algorithm is implemented in the HaRTES switch. For this evaluation, a network consisting of three switches and three nodes is considered. The network topology is depicted in Figure 5.13.

In this experiment the capacity of the links is 100Mbps. Currently, the HaRTES switch implementation uses netFPGA boards with four ports per switch, that are used for links. According to the extensive measurements reported in [104], 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 allocate $700\mu s$ for the synchronous window. Moreover, we generate 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 the RM algorithm. Table 5.4 shows the message parameters

including the period (T), priority (P), source node (S) and destination node (Ds).

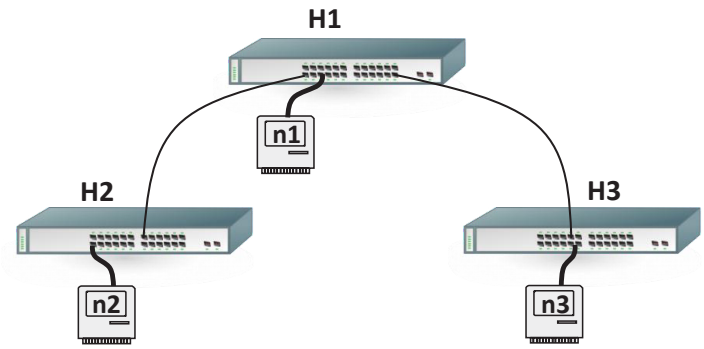

Figure 5.13: Experiment setup for the RBS method.

We measure the response time of the messages during 60 000 ECs. In Figure 5.14, 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 chapter. The computed response times (calculatedRT) are also illustrated in Figure 5.14 to show the difference of the measured and computed values. Analyzing Figure 5.14, we can see that the measured response times are lower than or equal to the response times predicted by the analysis. It can also be observed that the analysis embodies a certain degree of pessimism. 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.

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

Table 5.4: The messages' parameters for the RBS prototype experiment.

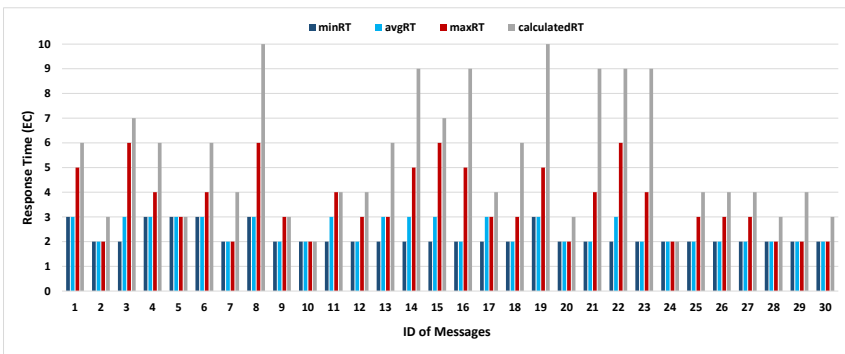

Figure 5.14: The minimum, average, maximum measured and computed response times.

We believe that the pessimism in the analysis stems from several sources. We discuss these sources herein, however a detailed study of the pessimism

and the solutions for them are not discussed in this chapter but remain as future work. The first source of the pessimism is related to the phasing of the messages. According to Equation 5.19, 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 Figure 5.13. In addition, assume that two messages, m_1 from node n2 to n3 and m_2 from node n1 to n3, share link in the link between switch H1 and switch H3. Although they have shared 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 Equation 5.8). However, the idle time may occur for smaller packets in reality.

5.4 Comparison of scheduling algorithms

This section compares the proposed scheduling methods in different perspectives. Firstly, it compares the methods with respect to the response time of messages crossing with the respective methods. It shows that the RBS method decreases response time of messages significantly compared with the DGS method. In the second evaluation, a network is defined with a set of messages that are generated randomly. Then, the response times of the messages are measured with simulation. The results of the simulation is also compared with the analysis. The third evaluation is using randomized network architectures.

In all evaluations in this section, we set the network capacity to $100Mbps$ and the hardware fabric latency of the switch (ϵ) is set to $3\mu s$. Moreover, 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.

5.4.1 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.

Scenario 1: Three-Switches Network

We considered a network comprising 3 switches along with 6 nodes, as illustrated in Figure 5.15. We generated 50 000 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 RM 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, the one with lowest priority, one medium priority and the one with highest priority. Then, we calculated the response time of the tagged messages according to the DGS and RBS methods. Note that we considered only schedulable sets. We computed the difference between the two response times for the tagged messages and we normalized the results with Equation 5.22.

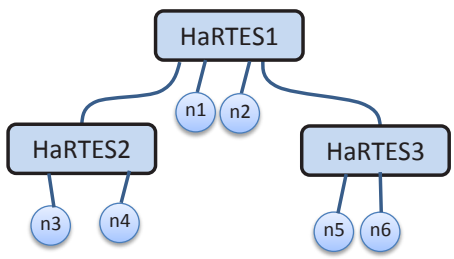

Figure 5.15: Three-switches network for comparing RBS and DGS methods.

$$Diff = \frac{RT^{DGS} - RT^{RBS}}{max(RT^{DGS}, RT^{RBS})} \times 100 \tag{5.22}$$

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 Equation 5.22. The negative value for $Diff$ shows that the response time for the tagged messages is smaller in the DGS method, whereas the positive value for $Diff$ indicates that the response time for the tagged messages is smaller in the RBS method. Also, the bigger value shows the bigger difference

between the response times. Figure 5.16 depicts the result of the evaluation for the network in Figure 5.15.

In Figure 5.16, the x-axis presents the percentage of the normalized difference between the response times in the RBS and 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 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.

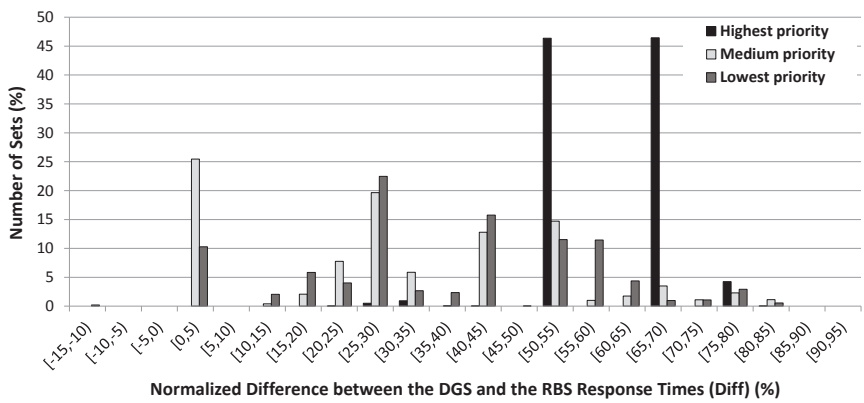

Figure 5.16: The difference between the response times in 3-switches network.

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 $[50\%, 55\%)$. The same happens for the lowest priority message. The response time in the RBS method is smaller compared to that of 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 behind this case where the DGS method gives better results using an example.

Let us assume that m_1 crosses switch H to reach node A in Figure 5.17, 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 3, we compute the response time of m_1 link-by-link. Thus, we calculate the response time in the input link considering m_a, m_b and m_c as interfering messages. Then, we compute the response time in the output link taking m_a and m_b as the interfering messages again, that might increase the response time of m_1. This is due to the fact that, in the RBS method, the message can be buffered in the last switch. However, when calculating the response time for the output link, m_a and m_b may not interfere as their effect was already accounted for in 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 as a future work.

Scenario 2: Seven-Switches Network

We extended the size of the network to seven switches with seven nodes, as depicted in Figure 5.18. We generated 50 000 sets, each of which containing

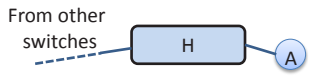

Figure 5.17: Special-case example in the DGS method.

30 messages. Similar to the previous evaluation, the messages are generated randomly as global and synchronous type. The properties of the messages are configured in the same way as the previous example. However, we set the EC size to $2ms$ and we allocate $1500\mu s$ for the synchronous window equally in all links. As the previous experiment, we tag three messages in these sets, one with the lowest priority, one medium priority and one with the highest priority. Then, we computed their response times according to the DGS and RBS methods. Again, we counted the sets that have a normalized difference of the response times within a certain value. Figure 5.19 illustrates this evaluation.

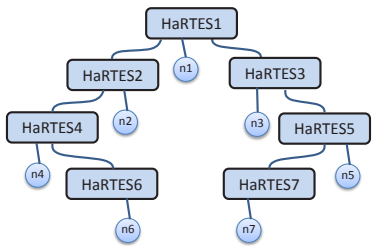

Figure 5.18: Seven-switches network for comparing RBS and DGS methods.

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 now have 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 occurrences of buffering is increasing 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 Figure 5.19 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.

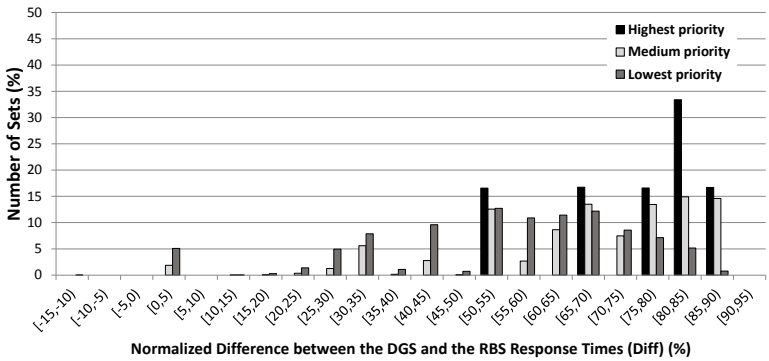

Figure 5.19: The difference between the response times in 7-Switches network.

5.4.2 Comparison based on simulation

For this evaluation a network example with three switches and six nodes is assumed, as shown in Figure 5.15. We generated a set of 20 messages with the following parameters. We selected the priorities of the messages based on the 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 mainly handle 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 the EC to $1ms$ and we allocate $0.7ms$ to the synchronous window in both 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

Table 5.5: The messages' respone time for the RBS and DGS methods.

Table 5.5 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 presented analysis 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 than or equal to the measured response times. The computed response times in the RBS method are always lower than the corresponding response times in the DGS method, except for message 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.

Moreover, according to the simulation results, the response times measured in the simulation in the RBS method are lower than or equal to the ones mea-

sured in the DGS method. The measured response times are equal for both RBS and DGS concerning the messages with lower priority, whereas when it comes to the higher priority messages then the RBS method provides shorter response time, e.g., m_5 and m_7.

5.4.3 Comparison on different networks

In this evaluation, 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 Figure 5.15 is 2, one is the master switch and the other is its children. Moreover, the number of hierarchy level in the network depicted in Figure 5.18 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 RM 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 using both the RBS and DGS methods. We measured the minimum, average and maximum response times of the messages in both methods. For each experiment (each generated network architecture), we focused on three messages, the highest priority, one medium priority, and the lowest priority. We derived the normalized difference of the measured response times in both methods for the three messages, with Equation 5.22. Figure 5.20 illustrates the normalized difference of the minimum measured response times of the messages in the RBS and DGS methods. The x-axis shows the normalized difference within a certain range of values, from 0% to 95%. Besides, the 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 have the normalized difference of minimum measured response times within $[80, 95]\%$ for the highest priority message.

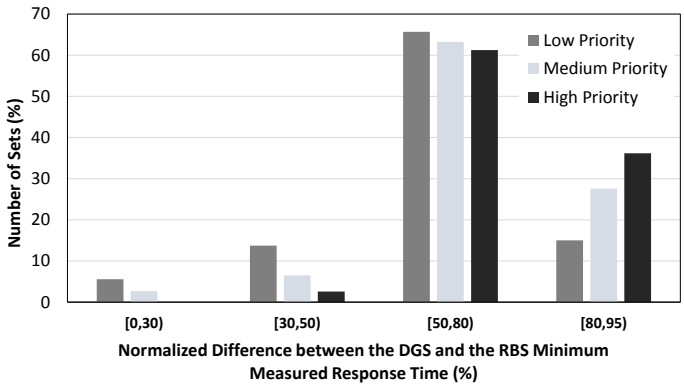

Figure 5.20: The difference between the minimum measured response times in RBS and DGS.

Figures 5.21 and 5.22 show the normalized differences of the average and maximum response times, respectively. The behavior follows the same way as the one described for Figure 5.20. Moreover, the pattern of the figures is similar to the ones depicted in Figures 5.16 and 5.19, 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 where the DGS method performs better than the RBS method.

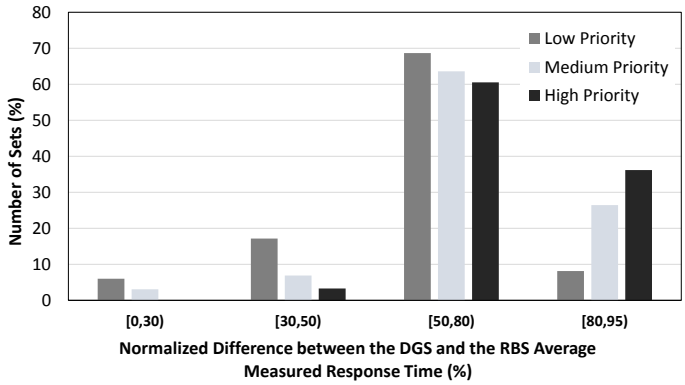

Figure 5.21: The difference between the average measured response times in RBS and DGS.

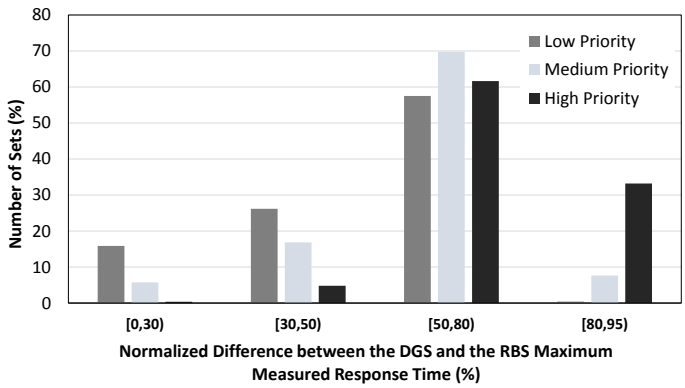

Figure 5.22: The difference between the maximum measured response times in RBS and DGS.

5.5 Conclusions

This chapter addressed the problem of supporting real-time multi-hop traffic in HaRTES architectures. Two scheduling methods are presented, namely DGS, based on buffering, and RBS, based on immediate forwarding. Then, the superiority of RBS is established, which allows a significant reduction of the

worst-case response times. Concerning the RBS versus DGS comparison, first their performance is compared by applying the response time analysis of each method on two different networks. It has been shown that the response time of the highest priority messages is always smaller in the RBS method and the response time of the lowest and medium priority messages is equal to or, in most cases, smaller than the case when applying the DGS method. Moreover, it has been shown that this difference increases for larger networks. We compared the response time of a set of messages in the RBS and DGS methods by means of simulation. Moreover, we generated random network architectures with random set of messages to validate our conclusion by comparing response times in different network architectures.

Chapter 6

MTU Configuration in FTT-SE and HaRTES

A prominent feature of the Ethernet-based protocols in general is that they can carry larger packet sizes compared to other communication technologies such as CAN. It has been shown that the MTU size of the Ethernet frames can have a big impact on the performance of the protocols, in particular on the latency of the packets [105]. In Ethernet-based protocols, each packet has a fixed size header to keep packet information such as the MAC address. Configuring the MTU size to a large value decrements the number of packets resulting from one message due to the fragmentation. Therefore, the overhead associated with the Ethernet packets decreases, which in turn increases the performance of the transmission. However, in cyclic-based switched Ethernet protocols, such as the FTT-SE and HaRTES architectures, a contradicting behavior occurs. Using a large MTU size in such architectures increases the idle time in each cycle of the transmission, hence it reduces the bandwidth efficiency. This contradicting behavior arises an optimization problem for designating the best MTU size. This chapter provides algorithms to select the MTU for individual messages such that the response time of the messages becomes as short as possible, leading to less resource utilization. This chapter presents the problem and solutions in the context of the multi-hop HaRTES architecture, then the results are extended to the multi-hop FTT-SE architecture.

6.1 MTU in the HaRTES networks

This section presents the problem of configuring the MTU for messages in the multi-hop HaRTES architecture where the messages are forwarded with the RBS method. Firstly, the problem of selecting the MTUs in the network is formalized along with a motivational example. Then, a search-based algorithm with two techniques to reduce the search space is presented. Finally, as the algorithm has exponential time-complexity, a heuristic algorithm is presented to select the MTUs in linear time-complexity.

6.1.1 Effects of MTU on the response time

In order to show the effects of configuring MTUs on the response time of messages, we use the response time analysis for the RBS method in the multi-hop HaRTES architecture presented in Chapter 5. The idle time equals to the maximum packet size in the worst-case. Configuring the MTU size to a large value increases the idle time, which in turn decreases the efficiency in terms of using the bandwidth wisely. This can also be seen in the response time analysis. When the MTU size of all messages increases, the maximum packet size PK_i increases, hence according to Equation 5.16, $Id_{i,l}$ grows. Therefore, the inflation factor $\alpha_{i,a,b}$ computed in Equation 5.15 decreases, considering a fixed EC and LW. As the interfering terms are inflated by $\alpha_{i,a,b}$ in Equation 5.17, the delays become larger. This results in larger response time $rt_{i,a,b}$, and accordingly larger RT_i. On the other hand, setting a large value for the MTU of messages decrements the number of packets due to the fragmentation. This decreases the overhead of the messages. According to the system model, C_i represents the total transmission time of m_i including the protocol overhead. Considering C_i^* as the actual data size, C_i can be rewritten in Equation 6.1, where O is the size of protocol overhead and nP_i is the number of packets that compose m_i.

$$C_i = C_i^* + nP_i \times O \tag{6.1}$$

Moreover, the number of packets after fragmentation of the data C_i^* is expressed in Equation 6.2.

$$nP_i = \left\lceil \frac{C_i^*}{MTU_i} \right\rceil \tag{6.2}$$

Then, the total transmission time C_i can be reformulated, which is shown in Equation 6.3.

$$C_i = C_i^* + \left\lceil \frac{C_i^*}{MTU_i} \right\rceil \times O \tag{6.3}$$

Looking at Equation 6.3, by increasing the MTU size, the total transmission time of the message C_i decreases. The same way the total transmission time of interfering messages C_j in Equation 5.19 decreases. Therefore, according to Equations 5.17 and 5.19, the interference of higher priority messages decreases, which in turn reduces the response time $rt_{i,a,b}$, and RT_i. Moreover, in our model an MTU can be configured for each message. Despite providing more flexibility for setting the MTUs, this model arises a third effect to deal with. Configuring a large MTU for the lower priority messages generates a bigger blocking term $B_{i,a,b}$ for the higher priority messages, according to Equation 5.20, which increases the response time of the higher priority messages. Therefore, configuring the MTU size for messages creates a contradicting behavior that may decrease or increase the response time of the messages.

6.1.2 Motivational example

Despite the above simple explanation about the effects of the MTU size, finding the best MTU size for messages is a nontrivial problem. We illustrate this through a simple example. Let us consider a network architecture consisting of two HaRTES switches along with four nodes. In this architecture, two nodes (N1 and N2) are connected to one switch, and the other two nodes (N3 and N4) are attached to a second switch. We illustrate the network example in Figure 6.1.

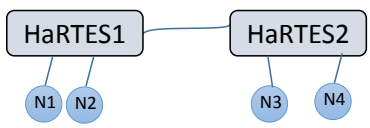

Figure 6.1: The network for motivational example.

In this example, we define two messages, where m_1 is transmitted from N1 to N3, while m_2 is sent from N2 to N4. Therefore, both messages cross two switches, and they share the link between two switches. The properties of the messages are shown in Table 6.1.

According to the Ethernet standard, the payload of each frame can not exceed 1500 Bytes, which is $120\mu s$ considering a $100Mbps$ network capac-

	Trans. Time (μs)	Period (EC)	Priority
m_1	800	5	High
m_2	800	8	Low

Table 6.1: Message properties for the example in MTU setting.

ity. Therefore, the maximum MTU size can be selected equal to $120\mu s$. Let us assume that the minimum possible value for the MTU size is $8\mu s$, that is 100 Bytes payload. In this example, for each message any value within $[8\mu s, 120\mu s]$ can be selected for the MTU size. We computed the response times of m_1 and m_2 according to the presented analysis, by choosing MTU sizes from the range. Note that we picked the MTUs with granularity of $1\mu s$, hence 112 possible MTU sizes were available for each message. Figure 6.2 shows the response time of m_1 based on the MTU sizes. In the figure, the x-axis and the y-axis show MTU_1 and MTU_2, respectively. Moreover, the z-axis shows the response time of m_1 in number of ECs. As it can be seen by decreasing both MTUs the response time of m_1 is decreasing.

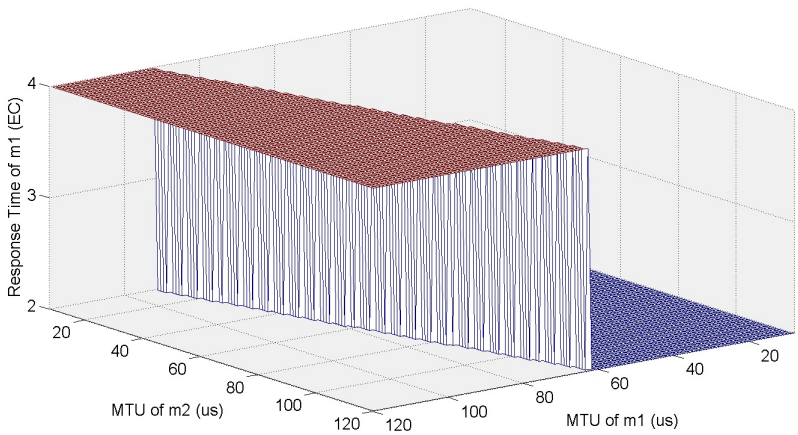

Figure 6.2: Response time for m_1 in the motivation example.

The effect of choosing MTU sizes on the response time of m_2 is illustrated in Figure 6.3. According to the figure, in order to have the shortest response time for m_2, MTU_1 must be within $[60, 120]\mu s$, and MTU_2 must be within $[18, 33]\mu s$. Moreover, to have the shortest response time for m_1, according

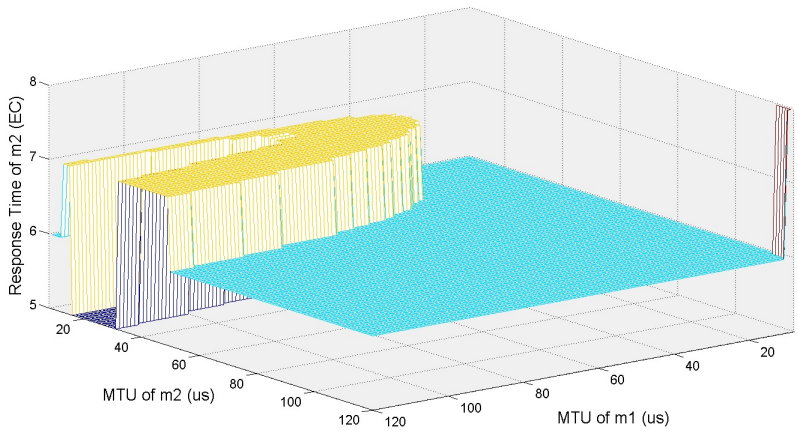

Figure 6.3: Response time for m_2 in the motivation example.

to Figure 6.2, $MTU_1 = [8, 60]\mu s$ and $MTU_2 = [8, 120]\mu s$. The intersection of the results gives the best MTU sizes, which are $MTU_1 = 60\mu s$ and $MTU_2 = [18, 33]\mu s$. By this setting the response times become $RT_1 = 2EC$ and $RT_2 = 5EC$. Note that by setting both MTUs to $120\mu s$, according to the figures, the response times become $RT_1 = 4EC$ and $RT_2 = 6EC$. It may happen that the intersection for one MTU is an empty set. For instance, suppose that in the mentioned example to have the shortest response time for m_1, $MTU_1 = [8, 59]\mu s$, and to have the shortest response time for m_2, $MTU_1 = [60, 120]\mu s$. In this case it is not possible to have the shortest response time for both messages by setting the MTUs. Therefore, we need to extend the range of MTU for at least one of the messages (e.g., the lower priority message) to gain one value from their intersection. By extending the range we can have $RT_2 = 6EC$ instead of $RT_2 = 5EC$, however it is still less than 7EC shown in Figure 6.3. In this example we defined two messages for simplicity of visualization. By incrementing the number of messages, finding the best MTUs becomes increasingly complex.

6.1.3 Optimization objective

Based on the discussion in the previous section, we can see that the problem is a multi-variable optimization problem. In order to minimize the response time of all messages, we define a cost function using RT_i and D_i. The optimization

objective is to minimize the cost function, while satisfying the constraint.

$$Minimize\ \frac{1}{N} \sum_{\forall i \in [1,N]} \zeta_i \times \frac{RT_i - D_i}{D_i} \tag{6.4}$$

$$Subject\ to\ MTU_{min} \leq MTU_i \leq MTU_{max}, \forall i \in [1, N]; \tag{6.5}$$

The main intention of the optimization objective is to decrease the response time of all messages. For this purpose we define a normalized difference between the response time and the deadline of the message, that should be minimized. In order to achieve a global optimization we define an average for the normalized difference of messages to be minimized. In addition, an importance factor (ζ_i) is multiplied to the normalized difference to indicate a weight for it. This way the normalized difference of a message with larger importance factor has more contribution to the average value. Minimizing the average value of normalized differences, considering the importance factor, means that the response time of the messages is as low as possible in relation to their respective deadlines. The importance factor will be explained later in this section.

Above, MTU_{min} and MTU_{max} are the boundaries of selecting MTU_i, which are defined according to the protocol specification. Assuming $\zeta_i = 1$ for now, the cost function of a schedulable system returns a value within $(-1, 0]$. The bigger negative value, i.e., the minimum cost function, shows the shorter response time for all messages. Note that the value cannot reach -1 as the response time cannot be zero. However, the cost function may become the same value for several different settings. Let us consider two scenarios each with a set of two messages m_1 and m_2. The messages have deadlines of 10EC and 20EC, respectively. In Scenario I, we set the MTUs such that the response time of the messages become 5EC for m_1, and 10EC for m_2, hence the cost function becomes -0.5. Now, in Scenario II, assume that we change the MTUs such that the response times become 6EC and 8EC for m_1 and m_2, respectively. Again, the cost function becomes -0.5. The scenarios are sketched in Table 6.2. In order to decide which MTU settings to select, the importance factor ζ_i is defined. The value of ζ_i is selected within $(0, 1]$, where a higher value shows the high importance of having a short response time for m_i.

In the same example let us assume that having a shorter response time for m_1 is more important than having a short response time for m_2. Therefore, we assign $\zeta_1 = 1$ and $\zeta_2 = 0.5$. According to the cost function presented in Equation 6.4, the cost function in Scenario I is -0.375, while in Scenario II it

	Scenario I			Scenario II		
	ζ	D	RT	ζ	D	RT
m_1	1	10	5	1	10	6
m_2	1	20	10	1	20	8
CostFunc	-0.5			-0.5		

Table 6.2: Example scenarios I and II

is -0.35. Therefore, Scenario I has the minimum cost function, that means the scenario in which the response time of m_1 is 5EC will be selected. In the cost function, the response time is a function of MTU_i. Moreover, the response time has a ceiling operation (Equation 5.19). Therefore, the optimization problem is a multi-variable nonlinear problem. One solution is to approximate the response time in the cost function by removing the ceiling operator. However, it makes the analysis very pessimistic, resulting in a not very accurate solution. In order to achieve an optimum solution in these types of problems, branch and bound techniques based on searching of a tree are normally used.

6.1.4 Search-based algorithm

As it is mentioned before, the maximum and minimum values for the MTUs are defined according to the protocol specification, where the range of MTUs is presented by $[MTU_{min}, MTU_{max}]$. The granularity in the range can be $1\mu s$, as the MTU size is presented in microseconds. This can result in a huge search space. However, later in this section, we propose two techniques to reduce that. The main concept of the algorithm is to check all combinations of MTUs, and to find the one that provides the lowest response time for the messages. In order to check the MTU combinations we generate a search tree. We start with the highest priority message in the set as the first node of the tree. Then, we draw one possible MTU for the message as a branch of the node. We pick the branch and assign the next node, which is the second highest priority message in the set. We continue until the last message, i.e., the lowest priority message in the set. In this state, we picked one MTU per message, i.e., one MTU combination is extracted. We compute the response time for the messages with the extracted MTU combination, followed by calculating the cost function. Then, we go one node back and select another branch for the node. Whenever all branches for the node are examined, we go back one node more to examine its branches. When all MTU combinations are examined, the MTU combination that gives

the lowest cost function is the best MTU combination.

We show this procedure in an example with two messages, m_1 and m_2. The tree search of the example is shown in Figure 6.4. Suppose that 3 MTUs for m_1 exist, which are $[MTU_1^1, MTU_1^2, MTU_1^3]$, and 2 MTUs for m_2, which are $[MTU_2^1, MTU_2^2]$. The algorithm starts with m_1 and the first MTU for that, i.e., MTU_1^1. Then, it selects m_2 and its first MTU, i.e., MTU_2^1. The first MTU combination is $[MTU_1^1, MTU_2^1]$, and based on them the response time analysis and the cost function calculation are performed. Then, the algorithm selects the second MTU for m_2, which makes the second MTU combination as $[MTU_1^1, MTU_2^2]$. The algorithm goes to m_1 node and selects the MTU_1^2 branch. This continues until the last MTU combination, which is $[MTU_1^3, MTU_2^2]$ in this example.

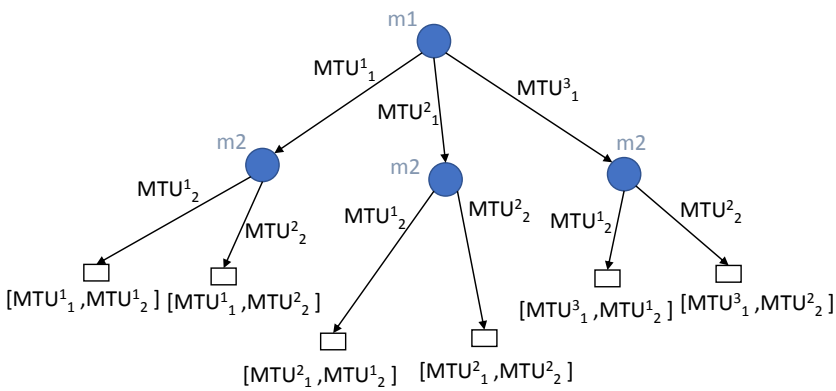

Figure 6.4: Tree of search - an example.

We present two techniques to reduce the search space. The first one is to increase the granularity of the MTU range, while the second one is to prune infeasible MTU combinations in the tree of search.

MTU range reduction

According to Equation 6.2, a range of MTUs gives the same number of packets due to the ceiling operation. Therefore, selecting one value from the range is enough, e.g., the maximum one. Algorithm 4 finds the minimum number of MTUs for N messages in the set, that essentially change the number of packets. In line 2 and the next of the algorithm, the minimum and maximum number

of packets for m_i is computed based on Equation 6.2. Then, the algorithm iterates for packet numbers between minimum and maximum, and it generates the possible MTUs for the messages. It stores the MTU in $actMTU$ if it was not generated in the previous iteration (line 9 and onwards). The complexity of the algorithm is $O(N \times nP^{max})$, where nP^{max} is the highest number of packets among the messages, i.e., $nP^{max} = \max_{\forall i \in [1,N]}\{nP_i^{max}\}$.

Algorithm 4 Find the possible MTUs.

1: **for** $i = 1$ to N **do**
2: $\quad nP_i^{min} = \left\lceil \frac{C_i^*}{MTU_{max}} \right\rceil$
3: $\quad nP_i^{max} = \left\lceil \frac{C_i^*}{MTU_{min}} \right\rceil$
4: $\quad x = 1$
5: $\quad$ **for** $j = nP_i^{min}$ to nP_i^{max} **do**
6: $\quad\quad M = \left\lceil \frac{C_i^*}{j} \right\rceil$
7: $\quad\quad$ **if** $j == nP_i^{min}$ **then**
8: $\quad\quad\quad actMTU_i^x = M$
9: $\quad\quad$ **else if** $j! = nP_i^{min} \&\& M! = actMTU_i^x$ **then**
10: $\quad\quad\quad x = x + 1$
11: $\quad\quad\quad actMTU_i^x = M$
12: $\quad\quad$ **end if**
13: $\quad$ **end for**
14: **end for**
15: **return** $actMTU$

Infeasible MTU combinations

The second technique is to check whether a branch in the tree has a feasible combination to continue searching. The infeasible branches are pruned from the search. We state the results in the form of a Lemma below.

Lemma 1. *If m_i does not meet its deadline by configuring the MTU of its lower priority messages to the minimum value, it does not meet the deadline with any other MTU values.*

Proof. The response time of m_i is calculated in Algorithm 3, followed Equation 5.17. Looking at the equation, the messages with priority lower than that of m_i only affect the blocking term $B_{i,a,b}$ and the switching delay $SD_{i,a,b}$. By

setting the MTU size of all lower priority messages to MTU_{min}, the maximum packet size used in Equation 5.20 is $PK_p = MTU_{min} + O$. Considering that the MTU size for messages with priority higher than that of m_i stays unchanged, the inflation factor calculated in Equation 5.15 stays unchanged in the blocking equation. Therefore, $B_{i,a,b}$ has its minimum value. Although the switching delay $SD_{i,a,b}$ in Equation 5.21 considers both high and low priority messages, by setting the lower priority messages to MTU_{min}, the results would only depend on the higher priority messages due to the max operation. Now if the response time computed in Equation 5.18 does not meet the deadline with the minimum blocking term, it cannot meet the deadline with other values of blocking. □

In the search tree, we select a branch of a node in the tree. According to Lemma 1, if by setting the MTU size of the further messages (the lower priority messages in the tree) to MTU_{min} the response time of the message in the node does not meet the deadline, we can safely prune the branch and continue with other branches. In order to implement the search tree, recursive search algorithms can be used (e.g., [158]). The complexity of the search algorithm in the worst-case is $O(M^N)$, where M is the maximum number of MTUs among the messages.

6.1.5 Heuristic algorithm

In this section, we propose a heuristic algorithm with a pseudo-polynomial time complexity based on the search-based algorithm. As we reduced the search space significantly by the proposed techniques, the heuristic becomes much faster, making it suitable to be used during run-time. The main intention is to select a branch in the tree which has the most likelihood of giving the minimum cost function, and to discard other branches from the search. Basically, the algorithm starts from the first message in the set, which is the highest priority, and draws its branches by selecting its MTU from MTU_{min} to MTU_{max}. The reason to start from MTU_{min} is that according to Lemma 1 if the message does not meet its deadline by configuring the lower priority messages to the minimum value, it does not meet the deadline with any other values. Therefore, the algorithm can check the schedulability of the message in its first attempt. Then, the algorithm sets the MTU of the lower priority messages to MTU_{min}, and computes the cost function. Moreover, the algorithm sets the MTU of the lower priority messages to MTU_{max} and computes the cost function. Then, it selects the branch with the lowest cost function to continue, and discards the

other branches.

The heuristic is presented in Algorithm 5. It iterates for N messages in the set, and for each message it iterates for $actMTU$ generated from Algorithm 4 (line 2). Note that $nMax_i$ is the index of the last MTU for m_i in $actMTU$. Then, it updates the MTU of m_i, while setting the MTUs of the lower priority messages to MTU_{min} (line 3). Line 4 of the algorithm checks if the branch is feasible to continue (Lemma 1). In case it is feasible, i.e., $chk = TRUE$, the algorithm computes the response time of messages (line 10), and calculates the cost function using Equation 6.4 in two cases where the MTU for the lower priority messages is set to MTU_{min}, and the case where the MTU of the lower priority messages is set to MTU_{max} in line 8. The MTU that causes the minimum cost function is stored in $finalMTU$ (line 13), then it continues for the next message in the loop. Note that if there are several MTUs resulting in the same cost function, the first one is picked by the algorithm (line 12).

Algorithm 5 Heuristic search algorithm in the HaRTES architecture

```
1: for i = 1 to N do
2:     for j = 1 to nMax_i do
3:         testMTU = update(j)
4:         chk = pruneCheck(testMTU)
5:         if chk = TRUE then
6:             for k = 1 to 2 do
7:                 if k == 2 then
8:                     testMTU = updateToMax(j)
9:                 end if
10:                rt = responseTime(testMTU)
11:                C = costFunction(rt)
12:                if (C < prevC) && (j! = 1) then
13:                    finalMTU = testMTU
14:                    prevC = C
15:                end if
16:            end for
17:        end if
18:    end for
19: end for
20: return finalMTU
```

The complexity of the algorithm in the worst-case is $O(N \times nMax \times 2)$. Note that responseTime function in line 10 has a pseudo-polynomial time

complexity. Therefore, the whole algorithm remains pseudo-polynomial.

According to Algorithm 5, the results of the cost function is always less or equal to either when all MTUs are set to MTU_{min} or MTU_{max}. The reason is that in the algorithm we examined the two cases and if there is no other combination better than them, the algorithm returns one of them depending upon the value of their cost function.

6.2 Experiments on the performance of algorithms

In this section we conduct three types of experiments. In the first part, we evaluate the proposed heuristic algorithm. In the second part, we generate small network configurations and we compare the two proposed algorithms, i.e., the heuristic and the search-based algorithms. Finally, in the last part, we show the improvement when reducing the search space.

6.2.1 Heuristic algorithm evaluation

In order to evaluate the algorithm, we generated 1000 random HaRTES architectures. In the generated architectures the number of hierarchy level is selected within [2, 4], the number of HaRTES switches is chosen within [3, 7], and the number of nodes connected to the switches is selected within [3, 30]. The number of hierarchy level shows the size of the architecture with respect to its depth, as the network topology is a tree. The nodes are randomly attached to the switches such that each switch has at least one node connected to it. Based on the size of the network, a set of messages is generated randomly. The number of messages in the set is selected within [10, 30]. The source and destination of the messages are chosen randomly among the nodes. The transmission time of messages is selected within [500, 1200]μs. Moreover, the period of the messages is set within [2, 30]ECs, and their deadlines are equal to the periods. The priority of the messages is assigned based on the RM algorithm. Furthermore, we set the importance factor $\zeta = 1$ for all messages. The network capacity is $100Mbps$, and the EC size is fixed to $2ms$. The transmission window within the EC is fixed in three values, 60%, 70% and 80% of the EC duration. For the generated messages we configured the MTU sizes using the proposed heuristic algorithm. Then, we compared the cost functions of the sets when setting the MTUs using the heuristic algorithm, and when the MTUs are maximum and minimum. We normalized the differences using Equation 6.6, where $\mathcal{C}_H$ is the cost function value when setting the MTUs using the heuristic algorithm,

and $\mathcal{C}_R$ is the cost function value when setting the MTUs to a reference. In this evaluation we have two references to compare with, i.e., MTU_{min} and MTU_{max}.

$$\sigma = \frac{\mathcal{C}_H - \mathcal{C}_R}{min(\mathcal{C}_H, \mathcal{C}_R)} \times 100 \tag{6.6}$$

Figure 6.5 shows the normalized difference between the MTUs selected by the heuristic algorithm and the references. The x-axis shows the transmission windows duration, whereas the y-axis shows the normalized difference (σ). The median of normalized difference of 1000 generated sets between the MTUs extracted by the heuristic and MTU_{max} is shown by Heuristic-Max bars. Moreover, the median of normalized difference of the sets between the MTUs extracted by the heuristic and MTU_{min} is shown by Heuristic-Min bars. As it can be seen in all three experiments, there is an improvement in the response time of the messages. For instance, when setting the window to 60% of the EC, the median of normalized differences for 1000 sets is around 20% compared with MTU_{max}. The vertical lines in Figure 6.5 illustrate the deviation of minimum and maximum calculated normalized difference. For instance, in the experiment where 60% of the EC is assigned for transmission, the maximum improvement is around 34% compared to setting the MTUs to MTU_{max}.

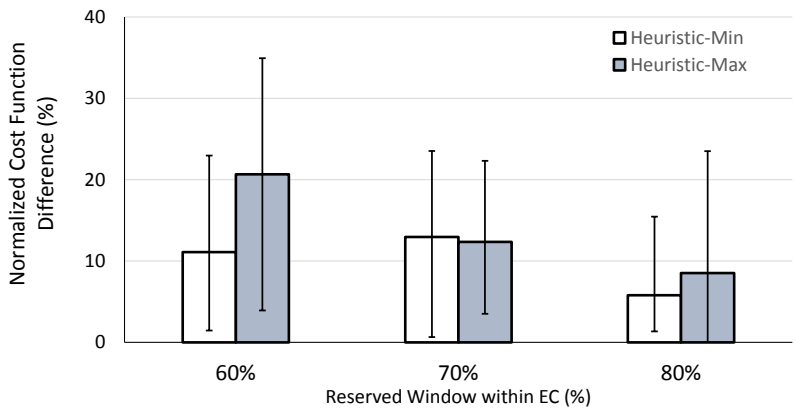

Figure 6.5: The heuristic vs. the references - varying the window size.

It should be noted that in some sets the normalized improvement was 0,

which means setting the MTUs to the maximum or minimum results in the same response time for the messages as setting the MTUs to the results extracted from the algorithm. However, we could not capture any negative difference in these experiments. Another interesting result is that by increasing the window size the normalized difference decreases. This is due to the fact that increasing the window size is diminishing the effect of the idle time (Equation 5.15). Therefore, by selecting the MTUs using the algorithm, we can use the resource more efficiently.

Moreover, we performed another experiment in which we investigated the effect of the number of messages on the improvements of our presented algorithm. Figure 6.6 shows the experiments of changing the number of messages from 10 to 30 in the sets. In this experiment, we fixed the network architecture to three HaRTES switches and six nodes. Moreover, we set the EC to $2ms$ and the transmission window to 60% of the EC. However, other properties are chosen randomly within the same ranges explained before.

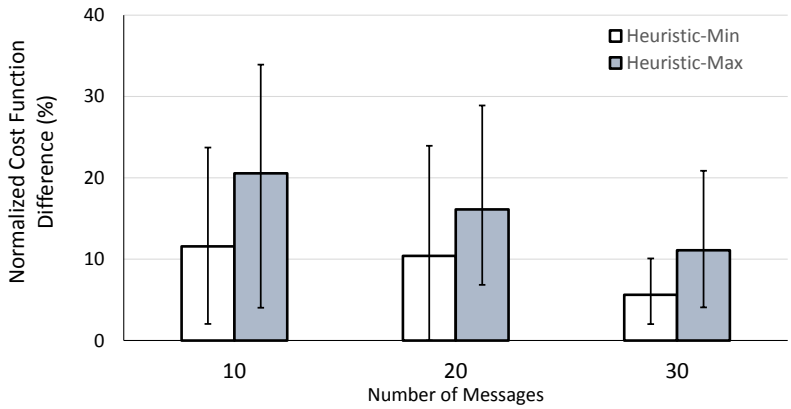

Figure 6.6: The heuristic vs. the references - varying the message number.

Figure 6.6 shows that we have an improvement in the response time of messages using the heuristic algorithm. The normalized difference reaches to 30% for the case of 10 messages in the network. By increasing the number of messages the normalized difference decreases. The reason is that the more messages that traverse the network, the more interference is generated. Thus, the system easily becomes unschedulable.

6.2.2 Search-based algorithm evaluation

In this experiment we compare the heuristic algorithm with the proposed search-based algorithm. We generated a network configuration consisting of two switches along with four nodes, where each switch has at least one node attached to it. Then, we generated 100 sets of messages randomly such that each set contains 3 messages. The transmission time and period of the messages are set randomly within $[900, 1000]\mu s$. The periods are selected within [3, 10]EC. Further, the priorities are assigned based on the RM algorithm. We fixed the EC duration to $1ms$, and we performed the experiment for four window sizes, i.e., 60%, 65%, 70% and 80% of the EC duration. We counted the schedulable sets among the 100 sets for each experiment, when using the heuristic and the search-based algorithms. Figure 6.7 illustrates the results, where the x-axis shows the window sizes, and the y-axis shows the percentage of schedulables sets. The figure shows that when we limit the transmission window, the schedulability of the systems reduces to around 23% when using the heuristic algorithm. However, with the same setting the search-based algorithm makes around 49% of the sets schedulable.

The experiment also shows that when the transmission window is not tight, the heuristic performs as good as the search-based algorithm to make the system schedulable. In general, when the transmission window is decreased, a lower number of messages can be transmitted, that results in a lower schedulability ratio. Therefore, the chance to find MTUs for messages to make them schedulable is lower. The chance for the optimum algorithm is higher as it checks all the search space. Note that, as the search-based algorithm has an exponential complexity, only a small network configuration can be evaluated in a reasonable time.

6.2.3 Search space evaluation

In this experiment we show that applying the search reduction techniques reduces the search space significantly. In order to perform this experiment, we generated a network configuration consisting of two HaRTES switches along with four nodes. The same as the previous experiments, the nodes are attached to the switches randomly. We generated 100 sets of messages, each containing two messages. The properties of the messages are chosen randomly within the same ranges described in the previous experiment. We performed the search-based algorithm to find the optimum MTUs for the messages in three different settings. In the first setting (SET1), we did not use the search

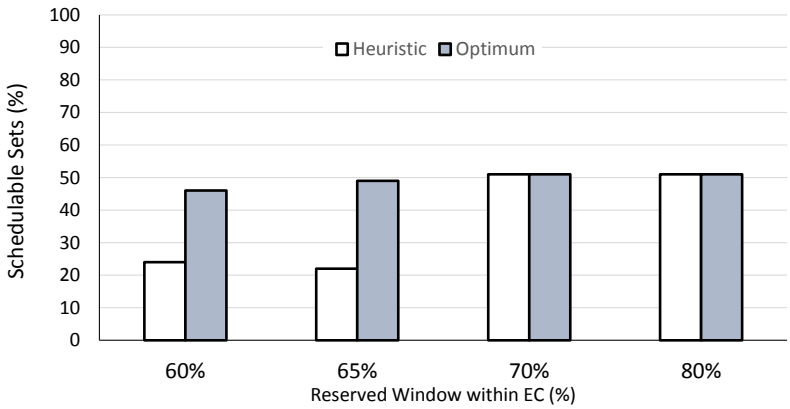

Figure 6.7: The heuristic algorithm vs. the search-based algorithm.

space reduction techniques, i.e. the search checked all possible MTUs within $[MTU_{min}, MTU_{max}]$ with $1\mu s$ granularity. Moreover, we did not prune the infeasible branches. In the second setting (SET2), we limited the MTUs for search using Algorithm 4, while in the third setting (SET3) we pruned infeasible branches on top of SET2. During the process of finding the MTUs, we measured the time that it takes to find the solution. Furthermore, we counted the number of MTU combinations that the algorithm searched. The results are shown in Table 6.3. This experiment shows that the reduction of MTUs in the range can significantly decrease the search space. The average number of MTU combinations reduced from 12769 to 1974 combinations in this evaluation, which equals to around 84% reduction in the search.

	SET1	SET2	SET3
Average time (s)	7.596	1.313	1.179
Average No. states	12769	2208	1974

Table 6.3: Search reduction techniques.

6.3 MTU in the FTT-SE networks

Similar effects on the HaRTES architecture can occur in the FTT-SE architecture as they use the same FTT transmission paradigm. Therefore, similar

algorithms can be used to find the most suitable MTUs for messages in the FTT-SE architecture, albeit with some adaptation.

6.3.1 Effects of MTU on the response time

In order to show the effects of changing MTUs for messages on their response times the response time analysis for the multi-hop FTT-SE architecture presented in Chapter 4 is considered. By increasing the MTU size of messages, according to Equation 4.4, the idle time increases as the MTU size affects directly on the packet size. Therefore, according to Equation 4.5, the bandwidth for transmission of the messages decreases, which in turn decreases the supply bound function in the analysis. Reducing the supply bound function increases the response time of the messages because of less network capacity provided for transmission. On the other hand, configuring the MTUs to the maximum value decrements the number of packets generated by fragmentation. Lower number of packets results in lower overhead per message, as it can be seen in Equation 6.7.

$$C_i = C_i^* + nP_i \times O \tag{6.7}$$

Therefore, compared with the case the number of packets are higher, the total transmission time of messages decreases when the MTUs are increased. According to Equation 4.1, by decreasing the total transmission time of the messages, the request bound function decreases, which makes the response time to decrease. Compared to the HaRTES architecture, the third effect from blocking by lower priority messages does not exist in the FTT-SE architecture. However, still the changes in the MTUs arises an optimization problem due to the contradicting effect on the request bound function and the supply bound function.

6.3.2 Optimization objective

The main aim is to decrease the response time of messages by configuring the size of MTUs. Therefore, the same objective function as shown in the HaRTES architecture is used in this case.

$$Minimize \ \frac{1}{N} \sum_{\forall i \in [1,N]} \zeta_i \times \frac{RT_i - D_i}{D_i} \tag{6.8}$$

$$Subject\ to\ MTU_{min} \leq MTU_i \leq MTU_{max}, \forall i \in [1, N]; \tag{6.9}$$

The response time of messages is computed by the analysis presented in Chapter 4. Moreover, MTU_{min} and MTU_{max} are the boundaries of selecting MTUs for messages, that can be set by the designers. The same weighting system applies in the FTT-SE case, where for each message a weight factor within $(0, 1]$ can be chosen. In this weighting system, 1 represents the highest importance factor. Therefore, the cost function can be within $(-1, 0]$, where the biggest negative value shows the shorter response time for all messages. Note that the cost function cannot be -1 as the response time cannot be zero.

6.3.3 Heuristic algorithm

The same search-based algorithm presented for the HaRTES architecture can be used for the FTT-SE architecture. However, for the HaRTES architecture two techniques to reduce the search space are presented. The first technique is to find an effective MTU range which can actually change the number of packets. Algorithm 4 gives a range for MTUs that can affect the number of packets, which can be used in the case of the FTT-SE architecture. The second technique is to remove the branches that end up to an infeasible combination. Lemma 1 proved that if configuring the MTUs of messages with lower priority than that of m_i to their minimum value does not make m_i schedulable, then no other MTU values can make m_i schedulable. This situation was possible due to the existing of blocking term in the response time analysis, which does not exist in the FTT-SE response time analysis. Therefore, the pruning technique cannot be used in the FTT-SE architecture. Therefore, a heuristic algorithm is presented to find the MTU of messages aiming at minimizing the cost function.

Algorithm 6 finds MTU size for messages such that the cost function becomes as much negative as possible. Firstly, the algorithm checks the cost function in two cases of configuring all MTUs to their maximum and minimum values, respectively. It should be noted that the *costFunction* function uses Equation 6.8 to find the cost function, by computing the response time of all messages. Then, the minimum of both cases is kept for the continuation. The algorithm initiates to check possible MTUs from the highest priority message in line 9. $nMax_i$ is the maximum index of MTU for m_i that is derived from Algorithm 4. Line 11 of the algorithm sets the j^{th} possible MTU for m_i and it sets the MTUs of all lower priority messages to their maximum value,

i.e., MTU_{max}. The cost function is computed according to the recent update for MTUs and the one which resulted in the minimum cost function is preserved. The MTU for the set is returned after examining all messages until the lowest priority message. As the algorithm checks the maximum and minimum values for the MTUs, the return MTUs from the algorithm is either shorter or equal to the response time of messages.

Algorithm 6 Heuristic search algorithm in the FTT-SE architecture

```
1: prevC = costFunction(MTUMAX)
2: C = costFunction(MTUMIN)
3: if C < preC then
4:     prevC = C
5:     finalMTU = MTUMIN
6: else
7:     finalMTU = MTUMAX
8: end if
9: for i = 1 to N do
10:    for j = 1 to nMax_i do
11:        testMTU = update(j)
12:        C = costFunction(testMTU)
13:        if C < prevC then
14:            prevC = C
15:            finalMTU = testMTU
16:        end if
17:    end for
18: end for
19: return finalMTU
```

6.3.4 Experiments on the performance of the algorithm

In order to examine the performance of the heuristic algorithm, we generated a network with three switches and six nodes. The network is depicted in Figure 6.8. Moreover, we generated a set of messages with random parameters as follows. The period of messages is set within a range of $[5, 25]$ECs and the priority of a message is assigned according to the RM algorithm. The transmission time of messages is selected within $[800, 1000]\mu s$. The network capacity is set to 100Mbps and the EC size is set to $1ms$. Within the EC the window for synchronous messages is set to different values in each experiment.

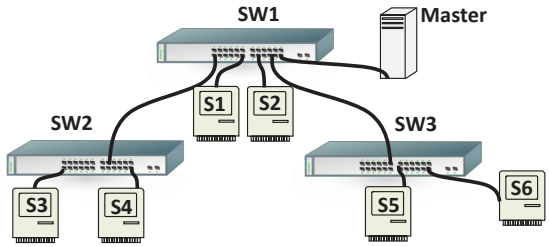

Figure 6.8: The network example for the FTT-SE architecture.

Figure 6.9 shows the results of the experiment where the transmission window is set to 70%, 80% and 90% of the EC. The number of messages in this experiment is decreasing for each set. For the evaluation, 1000 sets of messages are generated and for each set the cost function is computed in two cases. In the first case the MTU of messages is set to MTU_{max}, while in the second case the MTU of messages is set according to Algorithm 6. Figure 6.9 illustrates the normalized differences between the two cases. Note that the normalized difference is calculated based on Equation 6.6. The vertical lines in the figure show the minimum and maximum of the normalized difference and the bars show the average values. As it can be seen the normalized difference decreases by increasing the window size. This means that the algorithm finds better MTUs when the network is under stress with high response time for messages.

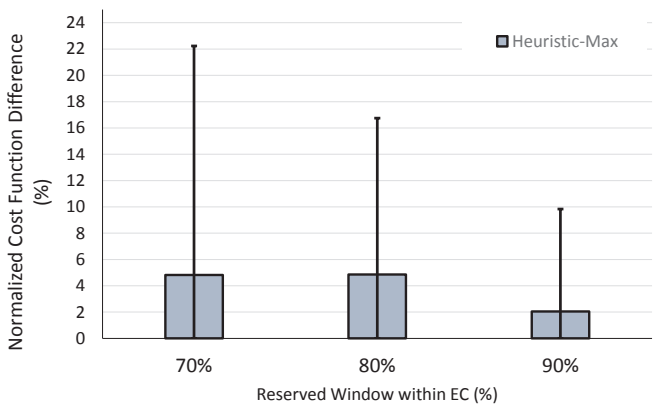

Figure 6.9: The heuristic vs. the reference - varying the window size.

In the second experiment the window size is fixed to 90% of the EC and number of messages in each set is changed from 5 to 15. It can be observed that by increasing the number of messages the success of the algorithm to find better MTUs decreases. This is because the algorithm misses more combinations for checking when the network becomes bigger with respect to number of messages. However, still the maximum normalized difference is only 6% for 15 messages in a set.

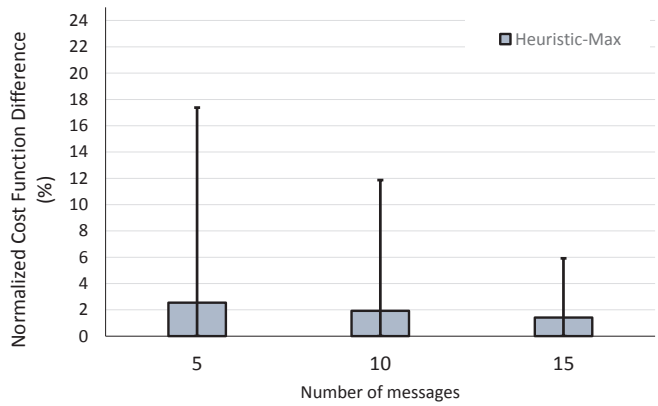

Figure 6.10: The heuristic vs. the reference - varying the message number.

6.4 Conclusions

This chapter focuses on improvements for the FTT-SE and HaRTES architectures. It shows that configuring the MTU sizes for messages in the architectures plays important role and affects directly on the response time of messages. However, the configuration is not trivial as there is a contradicting behavior in setting the MTU sizes. This chapter discussed this behavior both formally and by using a motivational example. Then two algorithms, one heuristic and one search-based, are presented to find the MTU for messages such that the response time of messages becomes as short as possible. Several experiments proved that the MTUs can be selected using the algorithms resulting in approximately 20% better response time.

Chapter 7

Dynamic Reconfiguration in FTT-SE and HaRTES

Previous chapters (Chapter 4, 5 and 6) presented solutions to obtain multi-hop communication over two architectures, the FTT-SE and the HaRTES. The chapters answered the first research question by evaluating the solutions looking at response time analysis and resource utilization. One of the main properties to preserve when extending original FTT-SE and HaRTES with multi-hop communication is the ability to handle dynamic reconfiguration, which is the second research question. This chapter addresses this functionality for both architectures. Dynamic reconfiguration as a response to changing in a dynamic environment is one of the prominent features in the FTT-SE and HaRTES architectures. An important aspect of dynamic reconfiguration in such networks is that the timeliness guarantee should be maintained during and after the reconfiguration process. This chapter presents two protocols for the FTT-SE network and one protocol for the HaRTES network, to perform such reconfiguration.

7.1 Dynamic reconfiguration in FTT-SE

In this section, two solutions extending the FTT-SE protocol of Chapter 3 are presented. The first solution is the multi-master FTT-SE architecture, where multiple master nodes, each of which is attached to a switch, collaboratively coordinate the traffic transmission. Whereas, in the second solution multiple master nodes coexist, but each of them controls the traffic in a group of

switches. The latter architecture performs better in terms of resource utilization. Therefore, in this section the challenges of providing adaptability and reconfiguration in the second solution (i.e., the hybrid architecture) are investigated. Two protocols to perform dynamic reconfiguration is presented.

As explained in Chapter 3, the scheduler in a master node scans the information in the SRDB database to schedule the traffic for the following ECs. Moreover, upon each TM reception by the slave nodes, they check whether they are the transmitter of any of the scheduled messages in that EC. These databases, i.e., SRDB database in the master node and NRDB database in the slave nodes, must be synchronized. This means that the messages and their parameters need to be identical in all databases. The online reconfiguration includes four basic steps. The first step is the negotiation between the slave nodes and the master node. The negotiation may refer to removing streams, adding streams, or changing the parameters of the streams. Reconfigurations are triggered by a slave node. In this work, we assume that the reconfiguration request is issued by the application layer (software) in the system. The second step is the admission control, to verify the feasibility of the proposed changes. In this step a utilization or response time analysis is used to check the feasibility of requested changes. The third step is the resource reservation in which the resources, after being distributed by the QoS management, are allocated to the message streams. The final step is the mode-change where the SRDB and NRDBs are updated. This transition is done gradually, yet in a bounded time, in order to provide a safe mode-change in the system.

In order to negotiate and perform the mode-change two real-time asynchronous messages are provided, one for sending a request from the slave node to the master node, and the other to send an update for the slave nodes. Note that the reconfiguration procedure is fully deterministic to achieve timeliness guarantee. Therefore, the mentioned steps are carried out in a bounded time including the request and update message transmission between slave nodes and the master node. This means that the response time of the assigned asynchronous messages for request and update is bounded and known.

The hybrid FTT-SE architecture comprises two types of traffic, including internal and external. The former one is local to a cluster, scheduled solely by one master. Thanks to the traffic isolation features of the FTT-SE protocol, which dedicates disjoint and independent windows to the internal and external traffic, the reconfiguration becomes a local problem, and in so being, the solutions previously developed for the single-switch FTT-SE networks [135] can still be used. However, the case changes radically when dealing with the external traffic. In this case the reconfiguration always involve all the clusters

of the network because the external transmission window is a shared resource managed cooperatively by all masters. Therefore, the steps involved in the reconfiguration must be instantiated in all master nodes in parallel and with a tight synchronization, to make sure that the SRDBs of each one are consistently updated over time and therefore that the scheduling decisions are correct and consistent. This section presents two protocols to carry out the reconfiguration. The first protocol, called *centralized reconfiguration*, uses one master node to perform all decisions regarding change requests. On the other hand, the second protocol, called *distributed reconfiguration*, is fully distributed in the sense that all master nodes process the change requests in parallel.

7.1.1 Centralized reconfiguration

The SRDBs of all master nodes contain the information of internal and external messages. Both internal and external messages are scheduled in parallel and independently. A correct behavior requires a network-wide consistent scheduling of external messages. This means that all master nodes must schedule exactly the same external messages in the same external windows. The parallel scheduling of external messages creates an end-to-end path through the involved switches that allow such messages to reach their destinations in one single cycle (actually, within the respective external traffic window). Attaining this behavior requires:

- The use of a deterministic scheduling algorithm;
- Synchronization among masters;
- A consistent representation of the external traffic in the SRDB.

The first two requirements are implicitly verified, since the FTT-SE protocol uses deterministic scheduling algorithms and the whole multi-hop framework depends on a proper synchronization. However, the third requirement implies that all changes to the external messages must be carried out on all master SRDBs and must take effect synchronously. As mentioned before, a reconfiguration involves several steps. Some of these steps, like admission control and QoS re-distribution, may encompass computationally intensive operations. Providing short response times to reconfiguration requests in those cases may require a considerable amount of processing power, much higher than the one necessary to carry out the regular master operations (e.g. scheduling and control of messages). In this scenario, it may be more resource-efficient having

a single node, embodied with a higher processing capacity, in charge of processing all the reconfiguration requests. The results are then communicated to the network master nodes, which only need to carry out a minimum extra processing. This architecture is designated *centralized*.

Without loss of generality, lets assume that the root master is the one responsible for deciding about the reconfiguration requests. The centralized method requires that all requests made by the slave nodes reach the root master. It is also necessary to allow the root master to communicate its decisions to the other masters. Thus, it is necessary to create a two-way channel between the root master and each one of the other masters, to convey the reconfiguration requests and replies. The global event sequence involved in a reconfiguration is depicted in Figure 7.1, where Slave 1 (S1), belonging to the cluster managed by Master 2 (M2), issues a reconfiguration request that also affects a message that is produced/consumed by Slaves 2 (S2) and 3 (S3). S2 also belongs to the cluster managed by M2, while S3 belongs to the cluster managed by M1.

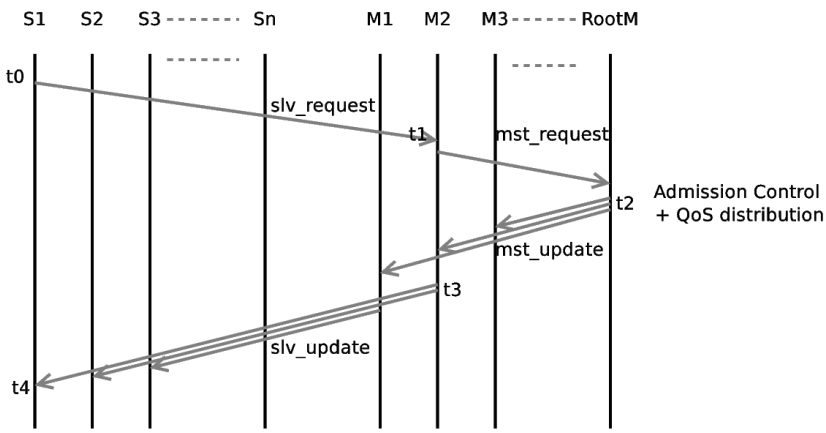

Figure 7.1: Event sequence for centralized reconfiguration.

Slave 1 requests a reconfiguration at time t_0, issuing a *slv_request* message to its cluster head (M2). M2 forwards the request to the root master (RootM), via a *mst_request*, at time t_1. RootM carries out the admission control and eventually the QoS redistribution. The results of these operations are then communicated to the other master node(s) at time t_2. If the change request is

denied, RootM sends a *mst_update* message to M2, notifying the decision, and no SRDB updates are required. Conversely, if the decision is positive, RootM sends a *mst_update* message to all master nodes, specifying which changes must be made to the respective SRDBs. For both cases the master nodes inform the slaves of the result of the reconfiguration, to synchronize the SRDBs and the NRDBs via *slv_update* messages.

Note that a single request may cause changes in several messages, due to the QoS redistribution. This may happen for example after deleting a message in a highly loaded system. The resource that becomes free may be distributed, by the QoS manager, among other messages that may take advantage of it. As a side effect, for systems with many external messages, *mst_update* and *slv_update* messages may have to be fragmented in several Ethernet frames.

Updating the SRDB and NRDB may take several ECs to complete. As discussed above, it is necessary that all the master nodes instantiate the updates at the same time, in order to schedule the external traffic consistently. Therefore, the *mst_update* messages encode the EC in which the changes should take effect. The RootM node is able to compute a safe upper bound to this value because the messages involved in the communication of its decisions (*mst_update* and *slv_update*) use asynchronous real-time (thus predictable) channels and RootM has global network knowledge.

7.1.2 Distributed reconfiguration

An alternative approach is to distribute the responsibility of the admission control and QoS management over all master nodes. Reconfiguration requests are forwarded to all master nodes, which concurrently evaluate their feasibility, compute the necessary updates to the SRDBs and communicate them to the respective slaves, to synchronize the NRDBs. This approach is possible because the master nodes have a consistent view of the shared resources (external messages are known by all master nodes and external windows have the same size in all clusters) and the admission control and QoS management algorithms are deterministic. Under these circumstances, despite operating quasi-independently, master nodes will reach consistent decisions. The only problem that remains is guaranteeing that the reconfiguration decisions are applied synchronously by all master nodes. Assuring this implies obtaining an upper bound to the execution time of the Admission Control and QoS tasks in all master nodes, in addition to an upper bound to the delivery of management messages. These constraints can also be met, since master nodes must have a real-time support (executive or RTOS), as they have to carry out sev-

eral real-time tasks (e.g., scheduling and TM dispatching), thus it is possible to bound the execution time of those algorithms. Additionally, as in the centralized approach, the reconfiguration is supported by real-time channels, thus the communication time can also be bounded. Fed with these two bounds, the master that receives a reconfiguration request can compute a time bound and encode it in the reconfiguration requests that it sends to its peers, permitting the synchronization of the instantiation of the reconfiguration results.

The global event sequence involved in a distributed reconfiguration is depicted in Figure 7.2. S1 sends a reconfiguration request to M2 at t_0. Subsequently M2 computes an upper bound to the processing time required by the reconfiguration request and forwards this information, together with the actual reconfiguration request, to all other master nodes (*mst_request* messages). Such requests are delivered until time t_2 and then processed. $U_P = t_3 - t_2$ represents the upper bound of the processing time of all master nodes. Then each master node notifies its slave nodes about the eventual changes to the NRDB, via the *slv_update* messages. Since these messages use a real-time channel, they are delivered on time by each master node, thus they take effect at the predefined time t_4.

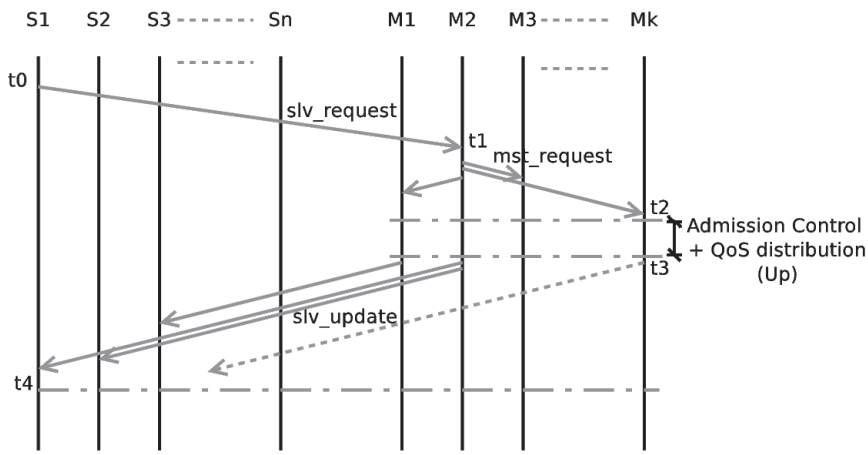

Figure 7.2: Event sequence for distributed reconfiguration.

It may happen that a master node receives two requests that are in conflict.

For instance, two requests may be issued by two nodes where only one of them can be accepted. In order to solve this issue the request time is included to the request message. Note that this time is presented by the EC number that the network is currently in. The request that was issued earlier will be considered in agreement by all master nodes. If they were issued at the same EC the request for changing a higher priority message will be considered by the master nodes.

7.1.3 Comparative evaluation

In this section, we evaluated the reconfiguration time which is divided into two parts: the computational time and the reconfiguration signaling time. The computational time is the time required to compute the response time of all messages in a set by one master node. The reconfiguration signaling time is the time required for negotiation and update steps. In order to evaluate these two parameters we considered two network examples, one small network with three switches and one larger network with five switches. The two networks are depicted in Figures 7.3 and 7.4.

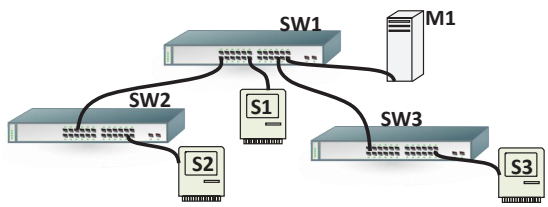

Figure 7.3: A network example with three switches.

In order to evaluate the computational time, we generated 1000 sets of messages randomly. The periods of the messages are selected within $[2, 10]EC$ and their priorities are assigned based on the RM algorithm, where 1 represents the highest priority. Note that, the messages can share a priority level in this evaluation when their periods are equal. Moreover, the transmission time of the messages are chosen within $[80, 123]\mu s$, where $123\mu s$ is the transmission time of a frame with 1500 Bytes payload. Moreover, the network capacity is set to $100Mbps$ and the EC size is set to $2ms$.

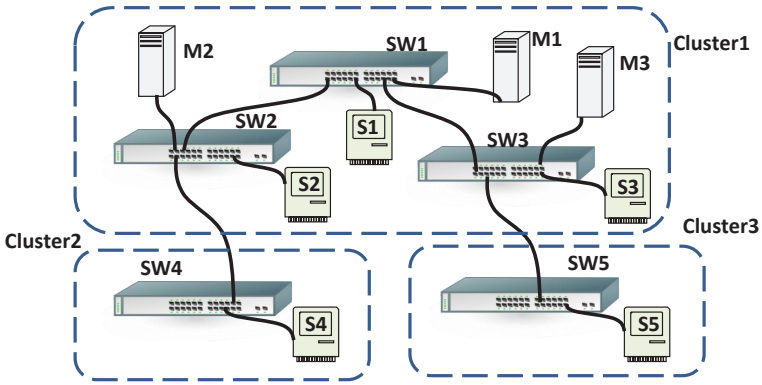

Figure 7.4: A network example with five switches.

As a master node in the network, we used an Atmel evaluation kit EVK1100 with an Atmel 32-bit AVR UC3A microcontroller. Remember that, this node is responsible to compute the response time for the messages. The response time calculation is done based on the analysis presented in Chapter 4. Note that in both the centralized and the distributed reconfiguration protocols, all master nodes should calculate the response time of all messages.

The computational time for the generated sets containing a different number of messages is presented in Table 7.1 and Table 7.2, for the networks depicted in Figure 7.3 and Figure 7.4, respectively.

No. of Msg	5	10	15	20	30	40
Min (μs)	2666	9717	21066	56566	60215	80854
Mean (μs)	2671	9724	21073	56571	60223	80857
Max (μs)	2677	9728	21077	56577	60225	80865

Table 7.1: Computational time in the small network example.

No. of Msg	5	10	15	20	30	40
Min (μs)	2677	9770	21408	57142	61527	81435
Mean (μs)	2682	9778	21412	57151	61535	81532
Max (μs)	2688	9781	21418	57153	61537	81604

Table 7.2: Computation time in the large network example.

As it can be seen in both tables, the computational time is increasing sharply by increasing the number of messages. Considering the EC size of $2ms$ for this evaluation, the average computational time for 10 messages in both the small and the large network is 2 ECs and it is increasing up to 41 ECs for 40 messages. This shows that using a centralized reconfiguration method is more efficient for the networks with a high number of messages as we can have a high capacity node to compute the reconfiguration parameters and the rest of the master nodes can be regular capacity nodes. Moreover, the size of network (number of clusters and nodes) does not significantly affect the computational time, as in both the small and the large network examples, for the same number of messages, the computational time is the same (2EC). In fact, the only part that changes in the response time calculation by increasing the size of a network is finding the interference in all routes, which is much faster than computing the response time in several iterations.

Concerning the reconfiguration signaling time, we assumed a change request from node S2 in Figure 7.3. Then, we measured the response time of the negotiation and update signaling, i.e., the response time of the request and update messages. As the reconfiguration messages are asynchronous, we filled up the asynchronous window in the EC with other asynchronous data messages. However, the reconfiguration messages have the priority of 1 (the highest), hence they are scheduled before the data messages. Note that the data messages are generated randomly with the same setting as in the computational evaluation. Table 7.3 shows the reconfiguration signaling time for the small network example considering different number of messages in the set. Note that the response time is presented in number of ECs.

Number of Messages	10	20	30	40	50
Centralized (EC)	4	4	4	4	4
Distributed (EC)	4	4	4	4	4

Table 7.3: The reconfiguration signaling time.

As it can be seen, increasing the number of data messages does not affect the reconfiguration signaling time. The reason is that the reconfiguration messages have the highest priority and their transmission times are very small, hence they are always scheduled before the data messages. Moreover, in both the centralized and the distributed method, the signaling time of reconfiguration is the same. This is because the requests, in both methods, are crossing only one hop.

Table 7.4 shows the reconfiguration signaling time for the network example depicted in Figure 7.4. By increasing the size of the network to 3 clusters, the centralized method requires more time for reconfiguration. This is due to passing the requests to the root master node and sending back the decision to the master nodes. However, in the distributed method, the master node sends the requests to all other master nodes and the decision is sent to the slave nodes directly.

Number of Messages	10	20	30	40	50
Centralized (EC)	8	8	8	8	8
Distributed (EC)	6	6	6	6	6

Table 7.4: The reconfiguration signaling time.

Although the reconfiguration signaling time is lower when using the distributed method, considering the computational time that has bigger effect, it is still more efficient to use the centralized method. Moreover, handling concurrent requests in the centralized method is much easier due to the serialization of the requests in that method.

7.2 Dynamic reconfiguration in HaRTES

In the multi-hop HaRTES architecture, reconfiguration requests are typically initiated by slave nodes. A request refers to adding or removing a message, or changing its parameters. Then, the request is given to an admission control module inside the HaRTES switch to decide whether it is feasible. Afterwards, the affected nodes and switches should be informed about the decision to change the system mode. As mentioned before, one key element of a reconfiguration protocol in a real-time application is to preserve the timeliness guarantee before, during and after the reconfiguration process. This leads to the following considerations:

- The duration of the reconfiguration process must be predictable, i.e., a request must be handled within a bounded time, either with a rejected or an accepted decision.
- The state of nodes and switches that are affected by a request must be changed to the new mode simultaneously and consistently to guarantee that the service is not disrupted.

In this section, we propose a protocol to perform a reconfiguration taking the above requirements into account. To do so, we provide required definitions beforehand.

7.2.1 Definitions

In order to support a large-scale network, we divide the network into several groups, which are called *Clusters*. A cluster is a group of HaRTES switches along with their slave nodes that have the same parent switch in the tree topology. The clusters do not have any shared switch or node and the parent switch and its nodes are also included into the cluster. This is unlike the cluster definition in the multi-hop FTT-SE architecture where the parents sub-network is excluded from the child cluster. In each cluster the parent switch is called a *Cluster Head*. One switch in the network is assigned with a specific role, which is called a *Governor*. The role of the governor will be described later in this section. The governor can be the HaRTES switch in the root of the tree architecture. However, depending on the load of requests the governor can be replaced by any other cluster head in the network. This means, if most of the global requests are occurring in a specific cluster, the governor can be selected from that cluster to decrease the number of request transmissions. Note that the cluster and governor selection is performed at the design phase and it is fixed during run-time. According to the previous definition the governor is also a cluster head. For instance, in Figure 7.5, H2, H4, H5 and their slave nodes are grouped as one cluster. Also, H1, H3 and their attached slave nodes are grouped as another cluster. In this example, H1 and H2 are cluster heads, while H1 is a governor as well.

We divide the reconfiguration requests that are issued by slave nodes into two categories. If a request is issued to add or remove a message where its source and destination nodes are within one cluster, the request is called a *Local* request. Otherwise, if a request concerns a message with a source node and a destination node in two different clusters, the request is called a *Global* request. For instance, in Figure 7.5, if a request is issued by node n7 to add a message to be sent from node n7 to node n3, it is a local request, whereas if the request asks to add a message transmitting from node n7 to node n5, it is a global request.

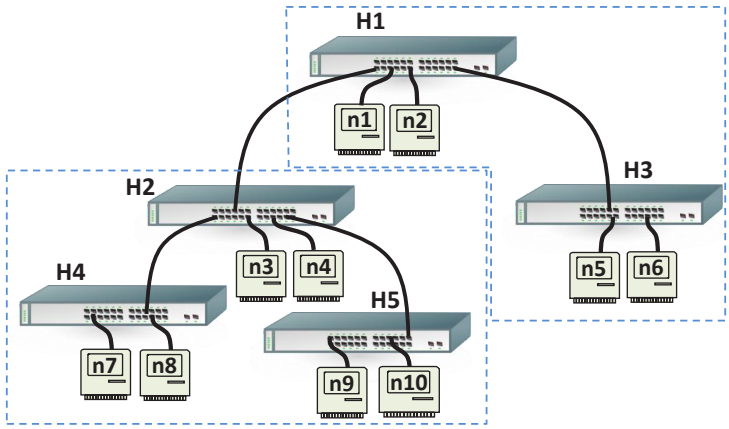

Figure 7.5: An example of HaRTES architecture.

7.2.2 Clustered reconfiguration protocol

In this protocol several HaRTES switches are collaborating in the reconfiguration process. The main intention of the protocol is that the local requests are handled by the cluster heads, whereas the global requests are managed by the governor. Next, we describe the reconfiguration protocol for the two categories of requests, separately. As a result of distributing the reconfiguration over several switches, a conflict may occur in the system due to parallel requests. In order to prevent this conflict a mechanism is designed which will be explained in this section, after describing the protocol in its normal operation.

Local Requests

The reconfiguration for local requests is done in five steps.

Step1-Negotiation: In this step a slave node issues a request for a change. The request is encoded in a message (*reqMsgL*) to be transmitted to the cluster head for which the slave node belongs to.

Step2-Decision: The cluster head receives the request and buffers it in a FIFO queue. The admission control of the cluster head picks the request in First-Come-First-Serve (FCFS) order, and checks its feasibility by computing the response time of all messages [24] considering the requested change. This requires consulting the messages database in the switch. After checking the feasibility of the change, the decision should be transmitted to the affected

nodes and switches.

Step3-Informing: In case of rejecting the request, the cluster head sends a reject message (*rejMsgL*) to the requesting node informing the impracticability of the request. However, if the request was feasible an accept message (*acpMsgL*) is sent to the requesting node and to the switch to which it is connected. This is essential so that the switch can update its internal database with the requested change, to be able to schedule accordingly.

Step4-Applying: The head should update its own database as it is responsible for checking the feasibility of requests according to the latest status of the network. In addition, the governor should be aware of the new changes in the network. The reason is that the governor is responsible for checking the feasibility of global requests. The global requests are targeting the messages crossing beyond a cluster. Therefore, the governor should have a global knowledge. For this purpose, the head sends an update message (*updMsg*) to the governor informing it about the introduced change. The updMsg is sent in both cases of accept or reject. The reason will be explained in the conflict prevention mechanism.

Step5-Mode Change: Changing to the new status should happen in all nodes and switches at the same time, i.e., at the same EC. The above mentioned messages used in the reconfiguration process are sporadic real-time, hence they are scheduled and transmitted within the Async Window. The priorities of these messages are selected as a design decision considering the importance of other data messages. Therefore, the arrival time of acpMsgL and updMsg is computable using the response time analysis. The latest EC in which both acpMsgL and updMsg are delivered is computed by the cluster head and encoded in the messages. This is the earliest EC for doing the mode change safely.

The reconfiguration control messages are real-time and the feasibility check of the requested changes is done with a real-time task executing under a real-time operating system. Thus, the whole reconfiguration process is bounded in time. Let us show the operation of the protocol with an example. Assume the network illustrated in Figure 7.5, where switch H1 is the governor, and H1 and H2 are cluster heads. Moreover, assume that node n7 requests adding a message. Figure 7.6 shows the sequence of the reconfiguration process. At time t_1 node n7 sends the request to the head, which is switch H2, through switch H4. Then, switch H2 carries out the feasibility check and sends an accept message to node n7 and switch H4 at time t_3. Note that switch H4 schedules the messages of node n7, hence its database should be updated about the messages in node n7. Moreover, an update message is sent to the governor, which is switch

H1, at the same time. The instant t_6 is the earliest safe time to apply the mode change, as both acpMsgL and updMsg are delivered. Computing this instant is also done in the cluster head after the feasibility check.

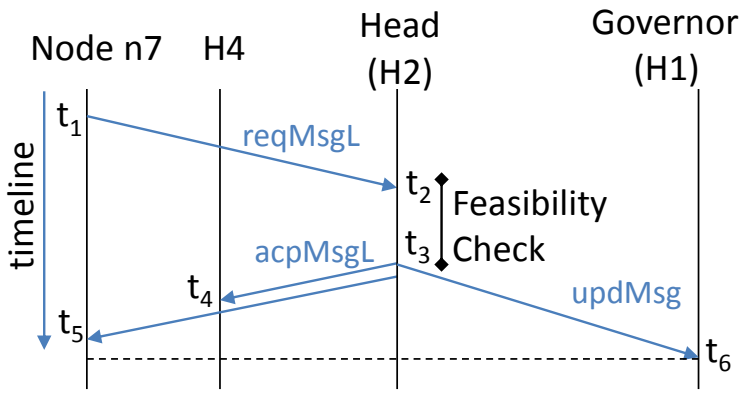

Figure 7.6: An example for a local request.

Global Requests

Global requests should be sent to the governor directly, instead of to the head. The steps of the reconfiguration are described below.

Step1-Negotiation: A global request is issued by a slave node and sent to the governor using a *reqMsgG* message.

Step2-Decision: The governor receives the request message and inserts it in a FIFO queue. In case of receiving several requests the governor deals with them in a FCFS order. The admission control inside the governor checks the feasibility of the request using the response time analysis given in [24].

Step3-Informing: The decision should be sent to the affected nodes and switches using a message. If the feasibility check is not confirmed, a *rejMsgG* is sent to the requested node. However, if the feasibility check is passed, an *acpMsgG* is sent to both the requesting node and the switch where the node is connected to.

Step4-Applying: The governor should update its database according to the new change. In addition, all other cluster heads should be aware of the change as they need the latest information for their feasibility check, that is done using an *infMsg* message. Although the heads only deal with the local requests, the

new changes may affect them. The reason is that the changes may affect the bandwidth in a cluster, hence the head of the cluster should be aware of how much bandwidth is available, to check the feasibility of a local request.

Step5-Mode Change: Similarly to the local request handling, the reconfiguration messages are sporadic real-time. Therefore, the instant to apply the mode change is determined by the governor after computing the latest arrival time of acpMsgG and infMsg.

Let us assume the network shown in Figure 7.5, where switch H1 is the governor. In this example, a global request is issued by node n5. The sequence of the process is illustrated in Figure 7.7. The request is sent to the governor at time t_1. The governor performs the feasibility check and sends an accept message to node n5 and switch H3 at time t_3. Further, the governor should inform the cluster heads using infMsg.

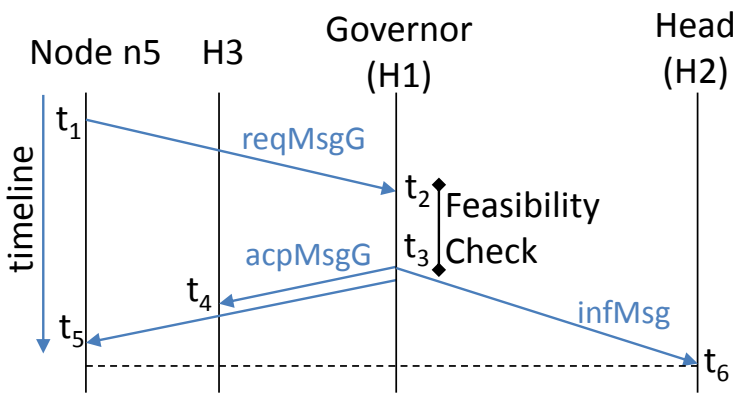

Figure 7.7: An example for a global request.

Conflict Prevention

According to the above proposed protocol, a head may accept a local request after checking its feasibility, and at the same time may receive an infMsg from the governor. It may occur that the infMsg has a new information that would make the local request infeasible. The same can also happen to the governor that receives an updMsg from a head that is inconsistent with a recently checked global request. This introduces a conflict, that needs to be prevented to ensure consistency in the network.

Let us show such a conflict with an example. Assume the network illustrated in Figure 7.5, where node n7 sends a request to add a message to be sent from node n7 to node n3. Moreover, assume that the link from switch H2 to node n3 has an available bandwidth for one message only. Therefore, the head (switch H2) accepts the request. At the same time, a global request is issued by node n5 to add a message to be transmitted from node n5 to node n3. As the request is global the governor (switch H1) deals with it. Again, the available bandwidth in the link to node n3 allows one message only, hence the governor accepts it since it does not know the other concurrent process, yet. In this situation, two messages are added causing an overload in the link to node n3.

In order to prevent the conflict, we present the following additional signaling for the proposed protocol. The main intention of the conflict prevention is to block the request processing of a cluster head when the governor is in the middle of the global request handling. Whenever a cluster head receives a request, it will check with the governor by sending an *askMsg* message whether it is in the middle of a global request process. If it is not, the governor sends back a message (*freeMsg*) to the cluster head showing that the governor is free. Then, the governor blocks any process until it receives the updMsg from the cluster head. However, if the governor is processing a global request while it receives the askMsg, it sends a message (*blkMsg*) to the head cluster informing it to block the process of the local request handling. The cluster head waits for the infMsg from the governor to continue the local request process with the updated database. During the time that the governor sends the infMsg until it receives the updMsg the process of global request is blocked. Note that in both cases of accept and reject, the updMsg and infMsg should be sent to ensure the continuation of the protocol.

Assume the network example illustrated in Figure 7.5. Node n7 sends a local request to the head (i.e., switch H2). The sequence when the governor is free is shown in Figure 7.8. The head sends an askMsg to switch H1, and at the same time starts to check the feasibility of the request. Switch H1 sends back a freeMsg showing that the governor is not processing any global request. Also, it blocks the global request processing. Note that the feasibility check could be finished earlier than receiving the freeMsg, however, the head waits for receiving the message before sending the acpMsgL and updMsg. The governor is unblocked after receiving the updMsg at time t_6.

The sequence for the case when the governor is in the process of a global request is illustrated in Figure 7.9. The head sends an askMsg to the governor at time t_2, however the governor is in the middle of processing a global request. Therefore, it sends back a blkMsg message informing the cluster head

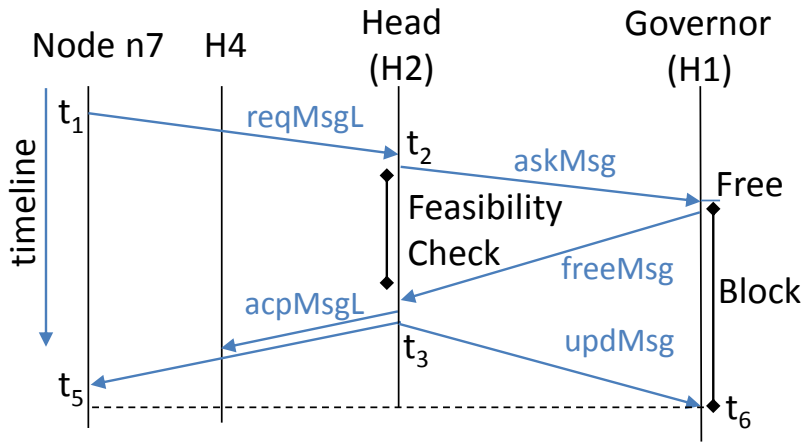

Figure 7.8: A conflict example when the governor is free.

to wait. When the governor is done, it sends the infMsg at time t_3 to inform the cluster head about the recent changes. At this point the head cluster checks the feasibility of the request with the updated information, that is finished at time t_4 by sending the updMsg and acpMsgL. During the time t_3 to t_5 the governor is blocked. There might be several global requests in the FIFO queue waiting for the feasibility check. However, the blkMsg and the infMsg are sent only for the current global request. Then, the governor keeps the other requests for processing after reception of the updMsg. This way the head is not blocked, possibly for an unbounded time, for all global requests in the FIFO queue to be served, instead it is only blocked for one global request.

7.2.3 Protocol evaluation

We conducted two sets of evaluations to show the performance of the clustered reconfiguration protocol. In the first set, we aimed at the admission control to see how much time it takes to perform a feasibility check for a request. Then, in the second set, we measured the time it takes for the reconfiguration process using simulations.

As mentioned before, the admission control is implemented inside the HaRTES switch. In this evaluation, we used a computer with a 4-core CPU running at 1.73GHz, considering one core as a processor of the admission control. The operating system is Ubuntu 14.04 and we made sure that the admission control

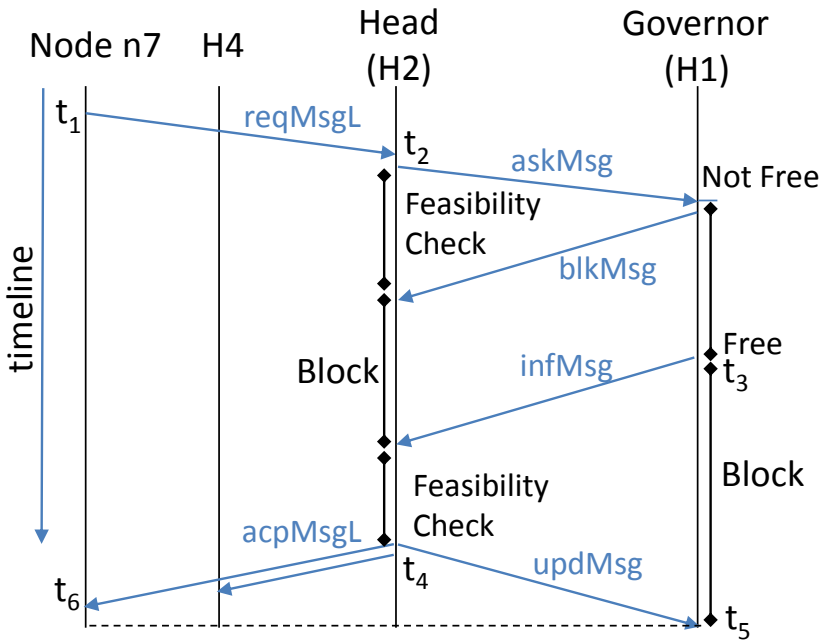

Figure 7.9: A conflict example when the governor is not free.

is not interrupted by any other process in Ubuntu. We implemented the response time analysis for the HaRTES architecture presented in Chapter 5. For the network architecture we considered the example depicted in Figure 7.5. We generated 100 000 sets of messages randomly. The periods of the messages are selected randomly within a range [5, 25]EC, and their priorities are assigned according to the RM algorithm. The transmission time of the messages is $120\mu s$, that is equal to 1500 bytes payload for the Ethernet frame. The source and destination nodes for the messages are set randomly considering that all messages are transmitted at least through two switches. Moreover, the network capacity is set to 100Mbps. We measured the time it takes for computing the response time of all messages in each set for three different network settings: (i) EC=$1ms$ and Async Win=$0.6ms$, (ii) EC=$1ms$ and Async Win=$0.8ms$, and (iii) EC=$2ms$ and Async Win=$1.6ms$. Moreover, we performed experiments for 10, 30, 50 and 70 messages in the set. The minimum, mean and maximum duration of time for the three settings are shown in tables 7.5, 7.6 and 7.7.

In general, by increasing the number of messages the time it takes to com-

No. of messages	min (μs)	mean (μs)	max (μs)	max (EC)
10	1.0	1.9	40.0	1
30	2.0	3.6	198.0	1
50	3.0	6.1	199.0	1
70	3.0	7.6	523.0	1

Table 7.5: Admission control when EC = $1ms$ & Async Win = $0.6ms$.

pute the response time increases. The important observation is that when we assign more bandwidth, i.e., larger EC and transmission window, the computation of the response time is faster. The reason is that using a larger transmission window results in a shorter response time for messages, that is achieved by a lower number of iterations in the response time calculation. In all cases for this example the feasibility check takes maximum 1 EC.

No. of messages	min (μs)	mean (μs)	max (μs)	max (EC)
10	1.0	1.8	40.0	1
30	2.0	4.1	43.0	1
50	2.0	5.5	46.0	1
70	3.0	7.7	48.0	1

Table 7.6: Admission control when EC = $1ms$ & Async Win = $0.8ms$.

No. of messages	min (μs)	mean (μs)	max (μs)	max (EC)
10	1.0	2.0	40.0	1
30	2.0	3.8	42.0	1
50	2.0	5.9	45.0	1
70	3.0	7.6	380.0	1

Table 7.7: Admission control when EC = $2ms$ & Async Win = $1.6ms$.

Concerning the reconfiguration process, we measured the time it takes based on simulation. We used a simulation tool, called SEtSim [153], which was initially developed to simulate the HaRTES architecture, and extended it with support for the clustered reconfiguration. We also set the feasibility check time to 1 EC according to the outcome of the previous evaluation. We considered the same network architecture as illustrated in Figure 7.5. We executed the simulation for 100 sets of messages, where the parameters of the messages are selected randomly within the same ranges as in the previous evaluation. We

performed the experiment when EC = $1ms$ and Async Win = $0.6ms$. Moreover, we considered 10, 30, 50 and 70 messages in the set, that share the bandwidth with the reconfiguration messages. The local and global requests are issued by nodes 5, 6, 7 and 8 with minimum inter-arrival between 12 EC to 25 EC. We executed each simulation for 1000 ECs. Table 7.8 shows the maximum time measured among the local and global requests. As it can be seen, the reconfiguration takes longer in networks with more messages due to sharing of the bandwidth. However, the maximum time we measured among all cases is 7 EC for a global request.

No. of messages	max - local (EC)	max - global (EC)
10	5	4
30	5	5
50	6	5
70	6	7

Table 7.8: Reconfiguration process when EC = $1ms$ & Async Win = $0.6ms$.

7.3 Model checking

There are many ways to validate a solution including simulation, experiment and formal verification. In the context of formal verification, model checking techniques are used to guarantee that the system performs as expected given an adequate model. Model checking techniques verify all possible states of the model of a system to check whether a specific property is satisfied. A well-known model checker tool is UPPAAL [159], suitable for applications of real-time systems [160, 161]. In this section, we present a number of UPPAAL models that we use to provide evidence for the correctness of the presented protocols. In the case of the hybrid FTT-SE architecture, the centralized reconfiguration performs better than the distributed reconfiguration. The centralized reconfiguration uses only one master node for the whole process, which is significantly simpler than the clustered reconfiguration. This is mainly due to the additional signaling for conflict prevention that does not exist in the centralized reconfiguration. In this section, model checking is presented for the clustered reconfiguration protocol and the centralized reconfiguration protocol.

7.3.1 UPPAAL basics

UPPAAL provides an ability to model a system as networks of timed automata. A timed automaton is a state machine with a local clock variable that evolves to measure the time progress. A timed automaton in UPPAAL is composed of states, called *locations*, and transactions across states, called *edges*. Moreover, abstract automata can be defined, called *templates*, that can be instantiated multiple times to create multiple similar automata, such as nodes in a distributed system. Figure 7.10 depicts an automaton in UPPAAL with two locations and one edge.

Figure 7.10: UPPAAL automaton example.

Over an edge, system variables can be updated, e.g., $x = 0$ in Figure 7.10. Moreover, two actions in two automata can be synchronized using synchronous channels. For example, the edge in Figure 7.10 is synchronized with an edge in another automaton via channel `sync!`. Whenever the edge with channel `sync!` fires, the edge with channel `sync?` is obliged to follow. A particular location is defined in the model where the system should leave it immediately, which is called *committed* location.

7.3.2 Model for the centralized reconfiguration

In order to model the centralized reconfiguration protocol, six templates are defined. Theses templates model different components in the protocol including the node which sends a request, the master of the cluster that the node belongs to, the root master and the buffers needed to collect the messages.

The `Node` template is illustrated in Figure 7.11, that starts in location `Initial`. The edge connected to the location activates a request to the master of the cluster via channel `slvReq[id]`. It should be noted that, similarly to the previous model, several nodes in the network are modeled by an input parameter (id). The model is suspended in location `Wait` until either the request gets accepted or rejected. These responses are activated via channels `slvRej[id]` or `alvAcp[id]`.

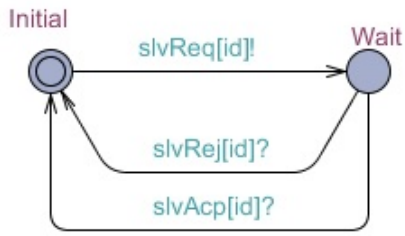

Figure 7.11: `Node` template.

A FIFO buffer collects the requests generated by the slave nodes. The model of the buffer in the master of cluster is depicted in Figure 7.12. Whenever a request is activated by a node, the edge in the buffer model is fired, which in turns update the buffer.

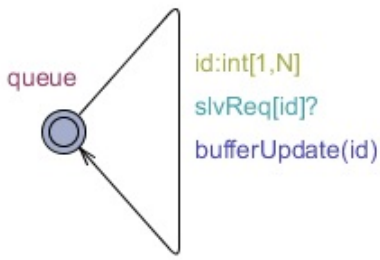

Figure 7.12: `clusterBuffer` template.

The template of the master of cluster is depicted in Figure 7.13. The master node starts from location `Initial`. Whenever the buffer which is collecting the requests has one member, i.e., `reqBuffer[first] != 0`, the edge is fired. The other locations until the beginning location are committed, which means their edges are activated immediately. In the second edge of the template the request coming from a node is sent to the root master node via channel `send`. After sending the request to the root master node the buffer removes the request with function `reqUpdate()`.

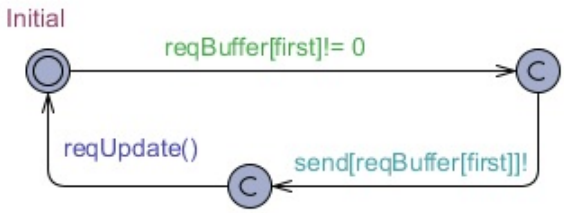

Figure 7.13: `clusterMaster` template.

Another buffer in the root master node collects the requests coming from the master of cluster. The `rootBuffer` template is depicted in Figure 7.14. The functionality of the template is similar to the buffer in the master of cluster, however the buffers are different.

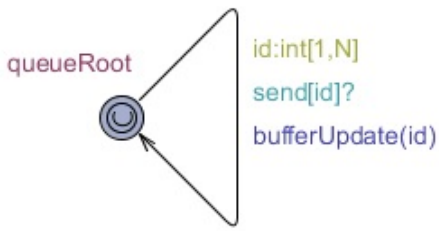

Figure 7.14: `rootBuffer` template.

The `rootMaster` template models the functionality of the root master node. Whenever a request is received by the master of cluster, the edge after location `Initial` is fired, if there is any request in the buffer. The feasibility check of the request is performed with the function `check(id)` and the answer returns as a boolean value in dec variable. In both cases of rejection or acceptance the response is sent to the node. Moreover, the root master informs other master nodes about the changes via channel `infMasters`. After serving the request the buffer is updated with the function `reqUpdate()`. In order to model the other master nodes that are informed by the root master node, the `masters` template is used, which is shown in Figure 7.16.

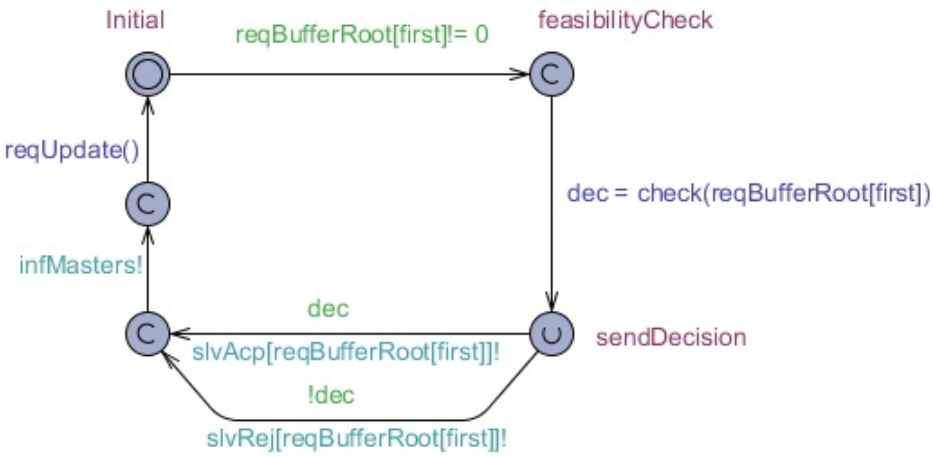

Figure 7.15: `rootMaster` template.

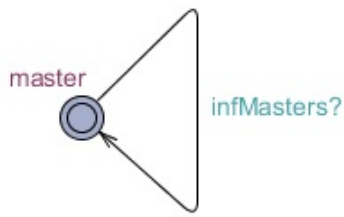

Figure 7.16: `masters` template.

The following property shows that whether the model is deadlock-free. We verified the property using the UPPAAL model checker and it was satisfied.

```
A[] not deadlock
```

Similarly to the previous model, we can verify that any request issued by a node will eventually be handled by either rejection or acceptance of the request. Therefore, the following property is checked for n number of nodes, which was satisfied.

```
Node(n).Wait—>Node(n).Initial
```

7.3.3 Model for the clustered reconfiguration

In order to model the clustered reconfiguration protocol, we defined 7 templates: (1) `localNode` that models the operation of slave nodes which issue local requests; (2) `Head` that models the head switch; (3) `Buffer` that models the FIFO queue to collect the requests; (4) `Switch` that models the HaRTES switch connected to the requesting node; (5) `Governor` that models a governor in the network; (6) `globalNode` to model the operation of slave nodes which issue global requests; (7) `govBuffer` that models a FIFO queue to collect the global requests in the governor switch. The `localNode` template models several nodes in the network, thus an input parameter of type integer (id) for identifying a specific node is defined in the model. This means that the `localNode` template represents several slave nodes by their identifier (id). Similarly, the `globalNode` represents several slave nodes by their index (j).

The `localNode` template is illustrated in Figure 7.17, that starts in location `Initial`. The edge to the next location activates a request to the head via channel `reqMsgL[id]`. The model is suspended in location `Wait` until either channel `rejMsgL[id]` or channel `acpMsgL[id]` is activated. These two channels show that the request is rejected or accepted, respectively. Then, the model starts again by sending another request.

Figure 7.17: `localNode` template.

We model a head switch with its FIFO buffer to collect the requests. The `Buffer` template is shown in Figure 7.18. Whenever a request is activated by a node, the edge in the buffer is fired by channel `reqMsgL[id]`. The request is buffered in a FIFO queue with function `bufferUpdate(id)` in the template. The buffer name is `reqBuffer`.

The cluster head is depicted in Figure 7.19. The head starts from location `Initial`. Whenever the buffer, that is collecting the requests, is not empty, i.e., `reqBuffer[first]!=0`, the edge to location `feasibilityCheck` is fired. Note that `first` represents the position at the top of the FIFO queue. Moreover, it triggers channel `askMsg` checking the governor whether it is pro-

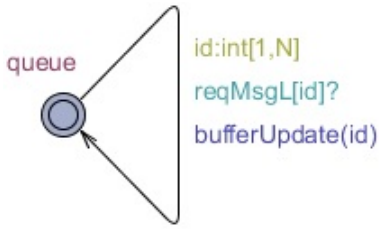

Figure 7.18: `Buffer` template.

cessing a global request. The model is suspended in the location until either channel `freeMsg` is triggered, which shows the governor is free, or channel `blkMsg` is triggered, that means the governor is processing and the head should be blocked. The model then stays in location `Block` until channel `infMsg` is triggered. Afterwards, the model decides on whether the request is accepted using function `check(reqBuffer[first])`. Note that in this model, the decision is taken by returning true or false to a variable `dec` in a non-deterministic manner. Based on the value of `dec`, the next edge from location `sendDecision` fires. In case the decision was reject, the edge guarded by `!dec` fires, which activates channel `rejMsgL[reqBuffer[first]]`, and in turn it triggers channel `rejMsgL[id]` in the `localNode` template. In contrast, if the decision was accept the edge guarded by `dec` is fired instead, which triggers an edge via channel `acpMsgL[reqBuffer[first]]`, which is `acpMsgL[id]` in the `localNode` template. In both accept and reject cases, the model moves to two committed locations towards location `Initial`. Over the last edges, channel `updMsg` is triggered to inform the governor, and the `reqBuffer` is updated to remove the request that is handled using function `reqUpdate()`.

As it was mentioned, the switch to which the slave node is connected should be aware of the accept decision. Therefore, the `Switch` template models the switch that should receive the `acpMsgL` as well, which is presented in Figure 7.20.

The global requests are handled by the governor. Therefore, we model the slave nodes that issue global requests, depicted in Figure 7.21. The template starts with an edge to trigger a request using channel `reqMsgG[j]`. The model is suspended in location `Wait` until it receives a decision from the governor with channel `govDec[j]`.

The buffer for the governor is modeled by the `govBuffer` template, depicted in Figure 7.22. In case of receiving a global request, which is triggered

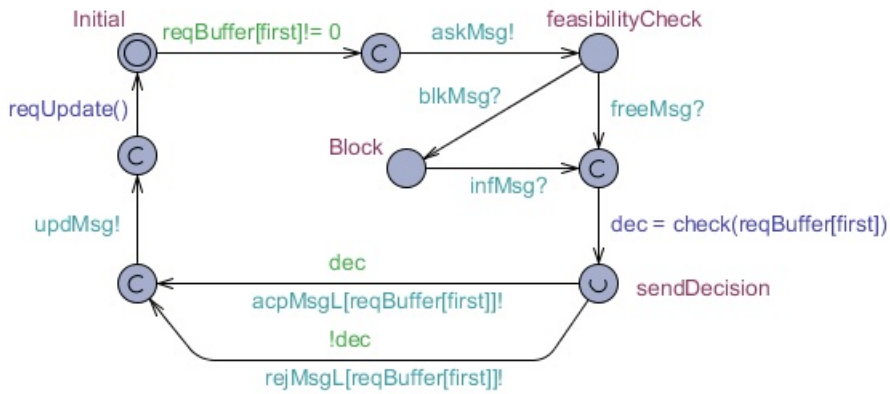

Figure 7.19: `Head` template.

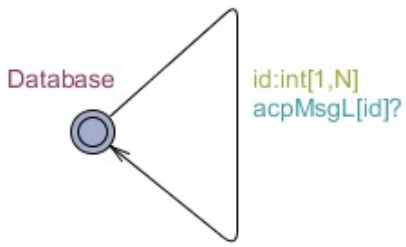

Figure 7.20: `Switch` template.

by channel `reqMsgG[j]`, the edge fires and the request is buffered into a queue by function `bufferUpdate(j)`.

The model for the governor is shown in Figure 7.23. The governor is activated if there is any request in its buffer. This edge fires by checking a guard `govRequest[first]!=0`, where `govRequest` is the buffer collecting the global requests and `first` represents the element at the top of the queue. It may occur that a cluster head triggers channel `askMsg`, to which the governor responds by triggering `blkMsg`. Then the channel `govDec` should be triggered to inform the node about the decision of the feasibility check. In the final phases, channel `infMsg` is triggered informing the cluster head about the change in governor, and the buffer to collect the global requests is updated

Figure 7.21: `globalNode` template.

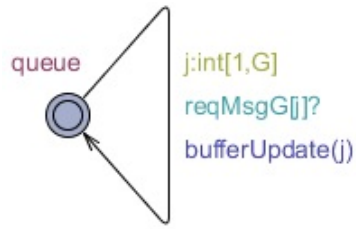

Figure 7.22: `govBuffer` template.

using a function `govUpdate()`. It may happen that when the model is in location `Initial`, channel `askMsg` is triggered. That means the cluster head is going to process a local request. Thus, the governor triggers channel `freeMsg` informing the head that it can continue. The governor is being blocked in location `Block` until it receives an updMsg from the head via channel `updMsg`.

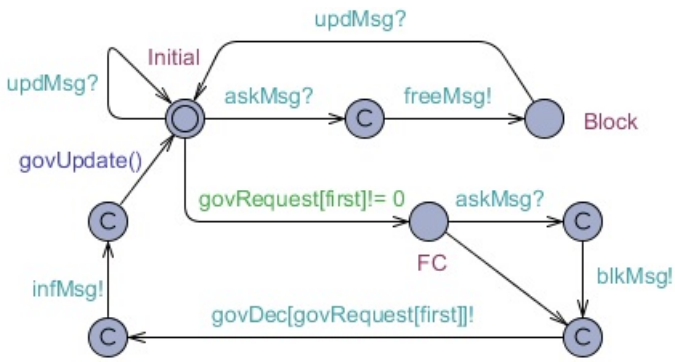

Figure 7.23: `Governor` template.

This model allows us to check if the protocol is deadlock-free. Therefore, we define the following property, which is satisfied for the model meaning that

the model does not get stuck in any location.

```
A[] not deadlock
```

Moreover, through model checking we can verify whether any request issued by a node will be eventually handled by either rejecting or accepting it. Therefore, we are interested in checking that when the model moves to the location `Wait` in the `localNode` template, it eventually moves back to the location `Initial`. We verify it with the following property.

```
localNode(1).Wait-->localNode(1).Initial
```

The property is satisfied for the model. Note that we showed the property for one node, i.e., `localNode(1)`, however we checked the model for five nodes sending requests, which are all satisfied. In addition, we checked whether the same property is satisfied for the nodes that issue global requests by the following property, which is satisfied as well.

```
globalNode(1).Wait-->globalNode(1).Init
```

7.4 Conclusions

Dynamic reconfiguration is a prominent feature of the FTT-SE and HaRTES architectures. This makes the protocol useful in dynamic environments where changes to the system state is demanded without taking the system offline. This chapter presented two reconfiguration protocols, one centralized and one distributed, to enable the reconfiguration process in a multi-hop FTT-SE network. Experiments showed that the centralized reconfiguration is more suitable in this context due to a lower complexity in terms of signaling and faster processing. Furthermore, a reconfiguration protocol, which is named the cluster reconfiguration, is presented for the multi-hop HaRTES network, Experiments showed the performance of the protocol. The model checking proved that the protocols are deadlock-free. In summary, this chapter addressed the second research question of the dissertation.

Chapter 8

Distributed Resource Reservation

Due to recent advances in computer-controlled functionality, the software in real-time systems has become overly complex. One way to deal with such complexity during the development is to split the system into several applications. The applications may then be developed by different teams or tier-1 suppliers independently. However, the applications still share the system resources such as processors and network bandwidth. In order to support isolation and independence among the applications after their deployment, resource reservation techniques [138, 44] are used. For instance, a portion of processor time can be allocated to a specific application using a hierarchical reservation framework [43, 42]. As a result, the timing requirements of each application can be verified independently of how applications are integrated. The applications can be integrated together without a need for analyzing the complete system as long as the timing behavior of each part has been independently verified. This can also provide flexibility during system development. There are several techniques to achieve temporal isolation among applications in the network domain. For instance, a multi-level hierarchical framework is presented by Santos *et al.* [63] and Iqbal *et al.* [141] that supports resource reservations in switched Ethernet networks. In this chapter the target is an end-to-end resource reservation that takes both network and computation into account. Therefore, it addresses the third (the last) research question. There are very few works that have addressed end-to-end resource reservations. One such work uses a distributed kernel to guarantee the end-to-end timeliness [146]. Another work

provides a distributed resource management system, called D-RES [149]. In the context of end-to-end resource reservations, another trend is towards using the resources efficiently by adapting to the run-time environment [145]. By temporally isolating various applications, such reservation techniques support online adaptation and reconfiguration of one application without affecting the rest of the applications in the system, given sufficient resources capacity.

8.1 End-to-end framework

This section presents a new end-to-end model that provides a framework in which system designers and system integrators can develop a system in isolation. The model of a real-time distributed embedded system that consists of multiple nodes is considered for this model. The nodes are connected to a single network that allows real-time communication. The nodes are assumed to be single-core processors executing real-time tasks scheduled according to fixed-priority preemptive scheduling. The system, denoted by $\mathcal{S}$, consists of L number of applications as shown in Equation 8.1. An application performs a specific function and is commonly distributed over several nodes. For example, the cruise control function in a car is one application.

$$\mathcal{S} = \{\mathcal{A}^{(1)}, ..., \mathcal{A}^{(L)}\} \tag{8.1}$$

8.1.1 Application and transactional models

An application consists of a set of M transactions as shown in Equation 8.2. A transaction is denoted by $\Gamma_i^{(h)}$, where i denotes its identifier and h denotes the identifier of the application to which it belongs. The transaction identifiers are unique within one application. However, two transactions belonging to different applications can have the same transaction identifiers.

$$\mathcal{A}^{(h)} = \{\Gamma_1^{(h)}, ..., \Gamma_M^{(h)}\} \tag{8.2}$$

A transaction consists of a chain (or sequence) of real-time tasks and messages which can have different activation patterns, i.e., a task or a message can be activated (released for execution or transmission) independently (e.g., either periodically by a clock or sporadically by an external event) or by its predecessor task or message. The tasks and messages in transaction $\Gamma_i^{(h)}$ are presented in Equation. 8.3.

$$\Gamma_i^{(h)} = \{\tau_{j,i,k}, ...; m_{i,k}, ...\} \tag{8.3}$$

The subscript, k, in $\tau_{j,i,k}$ represents the task identifier, whereas the first and second subscripts, j and i, specify the identifiers of the node and transaction to which this task belongs. We allow any two transactions to share the same task or message, provided that the transactions belong to the same application. Note that sharing of tasks or messages between two applications is not allowed. A transaction can include more than one task from the same node. The task identifiers are assumed to be unique within each node. For instance, $\tau_{1,2,2}$ is task 2 in transaction 2 and it executes in node 1, whereas $\tau_{2,2,2}$ is also task 2 in transaction 2, however it executes in node 2. Moreover, in Equation 8.3, $m_{i,k}$ represents message k in transaction i. Since we consider one network and assume the message identifiers to be unique, we do not use network identifier with the message.

It should be noted that a transaction cannot initiate or terminate with a message because a message must have a sender task and, at least, one receiver task. The transactional model is general in the sense that it can accommodate any number of tasks and messages that are in a chain. A transaction can also be composed of only one task, e.g., $\Gamma_1^{(1)} = \{\tau_{1,1,1}\}$. Such a transaction does not generate any message.

In order to understand the sharing of tasks among transactions, consider the example of a system that consists of three nodes and one network as shown in Figure 8.1. There are three tasks in Node 1; two tasks in Node 2; three tasks in Node 3; and two messages in the network. There is one application in the system that consists of two transactions denoted by $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ as depicted in Figure 8.1. Both of these transactions are initiated by the same task (i.e., the task with identifier 1) in Node 1. This task is denoted by $\tau_{1,1,1}$ in transaction $\Gamma_1^{(1)}$ and by $\tau_{1,2,1}$ in transaction $\Gamma_2^{(1)}$. A shared task among two or more transactions can be identified by comparing the first and third subscripts of each task in one transaction with the first and third subscripts of each task in every other transaction within the same application. The sequence of tasks and messages included in $\Gamma_1^{(1)}$ and $\Gamma_2^{(1)}$ are as follows.

$\Gamma_1^{(1)}$: $\tau_{1,1,1} \rightarrow \tau_{1,1,2} \rightarrow m_{1,1} \rightarrow \tau_{3,1,1} \rightarrow \tau_{3,1,2} \rightarrow \tau_{3,1,3}$.

$\Gamma_2^{(1)}$: $\tau_{1,2,1} \rightarrow \tau_{1,2,3} \rightarrow m_{2,2} \rightarrow \tau_{2,2,1} \rightarrow \tau_{2,2,2}$.

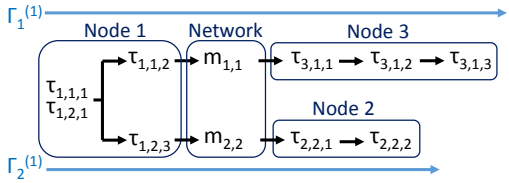

Figure 8.1: An example of two transactions with task sharing.

8.1.2 Task model

The task model complies with the model presented in Chapter 3. However, for this framework additional parameters are added to extend the model. The task is characterized by the following tuple.

$$\tau_{j,i,k} =< T_{j,i,k}, C_{j,i,k}, P_{j,i,k}, J_{j,i,k}, V_{j,i,k} > \tag{8.4}$$

A task can be activated periodically or sporadically. In the case of periodic activation, $T_{j,i,k}$ represents its activation period. Whereas, in the case of sporadic activation, $T_{j,i,k}$ represents the minimum inter-arrival time between any two consecutive activations of the task. If the task is activated by its predecessor, a precedence constraint is specified between the task and its predecessor. Such information is stored in $V_{j,i,k}$, which is null if the task is not activated by its predecessor. The release jitter of the task is specified by $J_{j,i,k}$. We assume that there is no synchronization among tasks.

8.1.3 Message model

The message model also complies with the model presented in Chapter 3. However, the model is adapted to follow the transaction model with additional parameters. A message in the end-to-end model is characterized by the following tuple.

$$m_{i,k} =< T_{i,k}, C_{i,k}, P_{i,k}, J_{i,k}, V_{i,k}, \mathcal{L}_{i,k} > \tag{8.5}$$

A message can be queued for transmission either periodically or sporadically. If the message is periodic, $T_{i,k}$ represents its period. Whereas, if the message is sporadic then $T_{i,k}$ specifies the minimum inter-arrival time that should elapse between the queuing of its any two consecutive instances. The release jitter of the message is identified by $J_{i,k}$. Moreover, the identifier of the task that sends the message $m_{i,k}$ is specified in $V_{i,k}$.

8.1.4 Resource model

Each application consumes a set of q resources. As shown in Equation 8.6, $R^{(h)}$ denotes the resource set for the h^{th} application, whereas r_q represents the q^{th} resource. The resource index, denoted by q, is unique within the system. Equation 8.6 includes node and communication resources. In a switched Ethernet network, one resource is defined per link.

$$R^{(h)} = \{r_1, ..., r_q\} \tag{8.6}$$

The total set of resources consumed by the system is obtained by performing the union operation on the resources consumed by all applications as shown in Equation 8.7.

$$\mathcal{R} = \bigcup_{\forall h \in \mathcal{S}} \{R^{(h)}\} \tag{8.7}$$

The resources are allocated to the applications using the periodic resource reservation policy. According to this policy, each application periodically receives a portion of bandwidth in each resource that it requires. We denote the reservation model of q resources for application $\mathcal{A}^{(h)}$ by $B^{(h)}$ as follows.

$$B^{(h)} =< \{\Pi_{r1}^{(h)}, ..., \Pi_{rq}^{(h)}\}, \{\Theta_{r1}^{(h)}, ..., \Theta_{rq}^{(h)}\}, \{\Psi_{r1}^{(h)}, ..., \Psi_{rq}^{(h)}\} > \tag{8.8}$$

Above, the replenishment period for resource r_q is denoted by $\Pi_{rq}^{(h)}$. The amount of budget to be replenished is denoted by $\Theta_{rq}^{(h)}$. In addition, we define a priority for the resource reservation which is represented by $\Psi_{rq}^{(h)}$ for resource r_q. The motivation behind the inclusion of the priority for each reservation is explained in the next subsection.

8.1.5 Node model

In order to support the resource reservation in nodes, we consider periodic servers [162]. We assume that there is one server per application per node that schedules the tasks assigned to it. Each node implements a two-level hierarchical scheduler, where both levels use the fixed-priority preemptive scheduling. The top-level scheduler, called the *global scheduler*, schedules the servers based on the priority of each reservation (Ψ). The second-level scheduler is called the *local scheduler* that schedules the tasks assigned to it. Figure 8.2

shows an example of one node that implements a two-level hierarchical scheduler. The global scheduler schedules two servers that schedule parts of two applications within the node. Each server has its own local scheduler to schedule the tasks that are assigned to it.

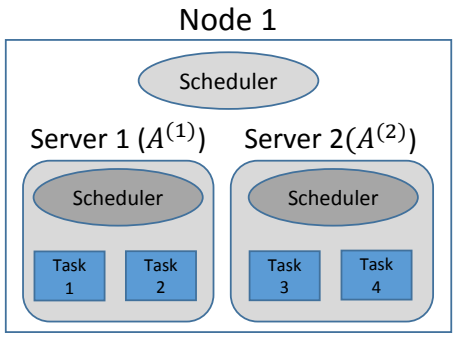

Figure 8.2: Example of a node with a two-level hierarchical scheduler.

8.1.6 Network model

For the network model any communication protocol that can support a resource reservation mechanism can be used. For instance, CAN with the server-based scheduling [163]. In the context of switched Ethernet networks, which is the main aim of this dissertation, AVB and HaRTES can be considered. However, as a proof of concept and due to the extension in the multi-hop communication in this work, the multi-hop HaRTES architecture with the RBS scheduling method has been chosen. Reservation in the HaRTES switch is more flexible than other technologies due to the possibility of implementing the hierarchical scheduling framework in each port. Furthermore, in the HaRTES architecture it is possible to reserve resources separately for each link. To implement the model two servers per application per network resource (i.e., per link) are developed, one for the Synchronous Window and another for the Asynchronous Window. The reason is that the synchronous and asynchronous transmissions should be isolated to keep the fundamental feature of the HaRTES architecture. Therefore, if there are two messages in one application crossing a link and from different types, they are served by two different servers.

8.1.7 End-to-end timing requirements

The end-to-end timing requirements on each transaction $\Gamma_i^{(h)}$ belonging to application $\mathcal{A}^{(h)}$ are specified as a set of constraints, denoted by $Cr_i^{(h)}$. These constraints include an end-to-end deadline, denoted by $D_i^{(h)}$; a data age constraint, denoted by $Age_i^{(h)}$; and a data reaction constraint, denoted by $Reac_i^{(h)}$. It is up to the user to specify one or more of these constraints on each transaction.

$$Cr_i^{(h)} = \{D_i^{(h)}, Age_i^{(h)}, Reac_i^{(h)}\} \tag{8.9}$$

8.1.8 Illustrative example

Consider an example of a distributed system with two applications, $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(2)}$. The system consists of two HaRTES switches that connect four nodes as shown in Figure 8.3.

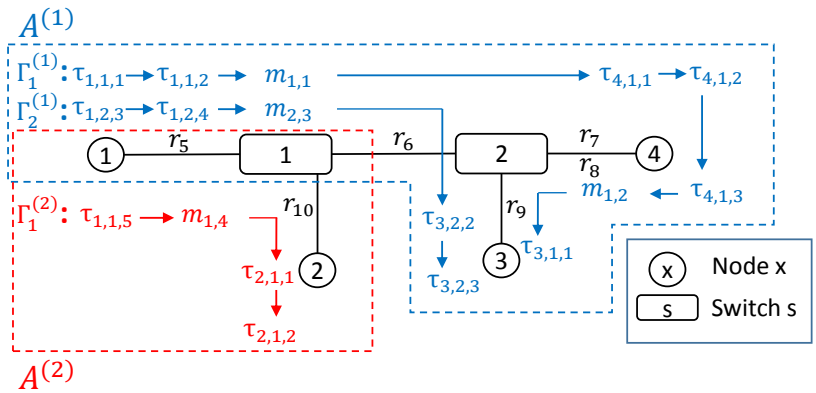

Figure 8.3: An example to illustrate the system model.

There are two transactions in $\mathcal{A}^{(1)}$ and one transaction in $\mathcal{A}^{(2)}$. The transactions for the two applications are as follows.

$\Gamma_1^{(1)} = \{\tau_{1,1,1}, \tau_{1,1,2}, m_{1,1}, \tau_{4,1,1}, \tau_{4,1,2}, \tau_{4,1,3}, m_{1,2}, \tau_{3,1,1}\}$

$\Gamma_2^{(1)} = \{\tau_{1,2,3}, \tau_{1,2,4}, m_{2,3}, \tau_{3,2,2}, \tau_{3,2,3}\}$

$\Gamma_1^{(2)} = \{\tau_{1,1,5}, m_{1,4}, \tau_{2,1,1}, \tau_{2,1,2}\}$

In Figure 8.3, the node resources are identified by r_1 to r_4, whereas the communication resources are identified by r_5 to r_{10}. Note that the resources r_7 and r_8 represent two directions of the network connection (i.e., one for each link). The two applications share resources in Node 1, denoted by r_1, and the link between Node 1 and Switch 1, denoted by r_5. The resources used by the two applications are represented as follows.

$R^{(1)} = \{r_1, r_3, r_4, r_5, r_6, r_7, r_8, r_9\}$

$R^{(2)} = \{r_1, r_2, r_5, r_{10}\}$

According to the reservation model, a reservation should be defined for each resource that an application uses. For instance, the reservation in resource r_4 for application $\mathcal{A}^{(1)}$ is $\{\Pi_{r4}^{(1)}, \Theta_{r4}^{(1)}, \Psi_{r4}^{(1)}\}$. This means that a server should be implemented in Node 4 with a period of $\Pi_{r4}^{(1)}$, a budget of $\Theta_{r4}^{(1)}$ and a priority of $\Psi_{r4}^{(1)}$. This server schedules three tasks, namely $\tau_{4,1,1}$, $\tau_{4,1,2}$ and $\tau_{4,1,3}$ as shown in Figure 8.3. Since r_1 is shared between the two applications, two servers should be implemented in Node 1. The first server, with parameters $\{\Pi_{r1}^{(1)}, \Theta_{r1}^{(1)}, \Psi_{r1}^{(1)}\}$, is responsible for scheduling the four tasks in $\mathcal{A}^{(1)}$, namely $\tau_{1,1,1}$, $\tau_{1,1,2}$, $\tau_{1,2,3}$ and $\tau_{1,2,4}$, that belong to Node 1. Whilst, the second server, with parameters $\{\Pi_{r1}^{(2)}, \Theta_{r1}^{(2)}, \Psi_{r1}^{(2)}\}$, schedules the only task, namely $\tau_{1,1,5}$, in $\mathcal{A}^{(2)}$ that belongs to Node 1.

8.2 Timing analysis with resource reservations

The system model (discussed in the previous section) strictly isolates the applications by resource reservations and by restricting sharing of tasks and messages among the applications. Therefore, schedulability of one application can be evaluated independently of the other applications. An application is said to be schedulable if all of its transactions are schedulable. That is, the end-to-end response times, data age and reaction delays in each transaction satisfy their corresponding constraints. In this section, first we present the response-time analyses for tasks and messages using our system model. In these analyses, we consider the implicit deadline model, i.e., the task and message deadlines are assumed to be equal to corresponding periods. Then we build upon these analyses and present the end-to-end response-time and delay analyses with resource reservations.

8.2.1 Response time analysis of tasks

We adapt the analysis presented in [139] as it is compatible with the presented model, e.g., the analysis is based on the assumption that the release jitter and deadline of each task are equal to or less than its period. The analysis is based on a *supply bound function (sbf)* and a *request bound function (rbf)*. The $sbf(t)$ is the *minimum* effective capacity that the resource provides within a time interval of $[0, t]$. On the other hand, the $rbf_i(t)$ is the *maximum* load generated by one task (τ_i) including the load of its higher priority tasks within the time interval of $[0, t]$. The earliest time where the $sbf(t)$ becomes equal to or larger than the $rbf_i(t)$ is the response time of τ_i. This method has been used in various models with different names for the supply and request bound functions.

The sbf for an application $\mathcal{A}^{(h)}$ in node j corresponding to the resource r_q is presented in Equation 8.10.

$$sbf_q^{(h)}(t) = \begin{cases} 0, & \text{if } t < \Delta \\ t - (\Delta + k \times (\Pi_{rq}^{(h)} - \Theta_{rq}^{(h)})), & \\ \text{if } \Delta + k \times \Pi_{rq}^{(h)} \leq t < \Delta + k \times \Pi_{rq}^{(h)} + \Theta_{rq}^{(h)} & \\ (k+1) \times \Theta_{rq}^{(h)}, & \\ \text{if } \Delta + k \times \Pi_{rq}^{(h)} + \Theta_{rq}^{(h)} \leq t < \Delta + (k+1) \times \Pi_{rq}^{(h)} & \end{cases} \tag{8.10}$$

where $k = \lfloor (t - \Delta)/\Pi_{rq}^{(h)} \rfloor$; while Δ is the worst-case latency of the server which is $2 \times (\Pi_{rq}^{(h)} - \Theta_{rq}^{(h)})$.

The rbf for the task $\tau_{j,i,k}$ that belongs to the application $\mathcal{A}^{(h)}$ in node j is presented in Equation 8.11. The tasks that are shared between transactions are excluded by checking the condition ($k \neq v$), i.e., if the task identifiers are not the same. If the interfering task $\tau_{j,p,v}$ is triggered by its predecessor, the period of its predecessor is extracted from $V_{j,p,v}$. The extracted period is then used instead of $T_{j,p,v}$.

$$rbf_{j,i,k}^{(h)}(t) = C_{j,i,k} + \sum_{\substack{\forall \tau_{j,p,v} \in \mathcal{A}^{(h)} \\ \wedge\ k \neq v \wedge\ P_{j,p,v} \leq P_{j,i,k}}} \left\lceil \frac{t + J_{j,p,v}}{T_{j,p,v}} \right\rceil C_{j,p,v} \tag{8.11}$$

The response time of task $\tau_{j,i,k}$, denoted by $RT_{j,i,k}$, is derived from Equation 8.12. The inequality should be evaluated for all t in the interval that is equal to the period of $\tau_{j,i,k}$.

$$RT_{j,i,k} = min(t > 0) : sbf_q^{(h)}(t) \geq rbf_{j,i,k}^{(h)}(t) \tag{8.12}$$

8.2.2 Response time analysis of messages

Among the existing analyses, the most suitable analysis to be applied to our model is the response-time analysis developed for the single-switch architecture [63], which is based on the Explicit Deadline Periodic (EDP) resource model [162]. However, the presented model considers that the messages traverse through multiple HaRTES switches with resource reservations. We present analysis for the multi-hop HaRTES architecture by adapting the existing analysis and using the sbf for the servers presented in [138]. A server is implemented for each network resource to schedule the messages that belong to one application, provided that the server is used during either the Synchronous or the Asynchronous Window, depending upon the activation patterns of the messages. The scheduling in the network link is performed in a hierarchical fashion, where the EC and its two windows constitute the top level while the servers within the windows represent the second level in the hierarchy. We provide the sbf for a message $m_{i,k}$ belonging to transaction i in application $\mathcal{A}^{(h)}$ that crosses link l_q, which is denoted by $sbf^{(h)}_{i,k,q}(t)$. Note that in the analysis we consider that messages are not larger than the maximum Ethernet size, hence there is no fragmentation of messages.

Since the messages cannot be preempted during their transmission, a portion of the budget in the server can be wasted. We define this wasted time as the *idle time*. Figure 8.4 shows a scenario in which a message cannot fit within the remaining budget in the first period of the server, hence it has to be sent after the replenishment of the server budget during the next period. The idle time is upper bounded by the maximum size of higher priority messages that cross the link l_q. The higher priority messages belong to the same application with the same activation pattern as that of $m_{i,k}$. The idle time is denoted by $Id^{(h)}_{i,k,q}$ and is calculated in Equation 8.13, where $AP(m_{i,k})$ presents the activation pattern for $m_{i,k}$.

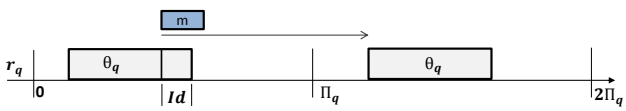

Figure 8.4: Example demonstrating the inserted idle time.

$$Id^{(h)}_{i,k,q} = \max_{\substack{\forall m_{z,p}\in\mathcal{A}^{(h)}\wedge\ P_{z,p}\leq P_{i,k}\wedge\ l_q\in\mathcal{L}_{z,p} \\ \wedge\ p\neq k\wedge\ AP(m_{z,p})=AP(m_{i,k})}} \{C_{z,p}\} \tag{8.13}$$

Therefore, in the calculation of the sbf, the idle time should be subtracted from the budget, i.e., $\Theta'^{(h)}_{rq} = \Theta^{(h)}_{rq} - Id^{(h)}_{i,k,q}$. The sbf can be computed as follows.

$$sbf^{(h)}_{i,k,q}(t) = \begin{cases} y.\Theta'^{(h)}_{rq} + max\{t - x - y.\Pi^{(h)}_{rq}, 0\}, \\ \qquad\qquad \text{if } t \geq \Pi^{(h)}_{rq} - \Theta'^{(h)}_{rq} \\ 0, \qquad\qquad \text{otherwise} \end{cases} \tag{8.14}$$

where $x = 2(\Pi^{(h)}_{rq} - \Theta'^{(h)}_{rq})$ and $y = \left\lfloor \frac{t-(\Pi^{(h)}_{rq} - \theta'^{(h)}_{rq})}{\Pi^{(h)}_{rq}} \right\rfloor$.

The rbf is defined as the maximum load generated by a message with respect to its critical instant. The critical instant, in this case, corresponds to releasing the message with all its higher priority messages at the same time. The interference and blocking received by a message in the HaRTES architecture has been derived in [24]. These interferences for $m_{i,k}$ are categorized as follows: (i) the interference from the higher and equal priority messages that share the link with the message, denoted by $I^{(h)}_{i,k,q}$, and (ii) the blocking from the lower priority messages that share the link with the message, denoted by $B^{(h)}_{i,k,q}$. The rbf of the message $m_{i,k}$ belonging to the application $\mathcal{A}^{(h)}$ crossing the link l_q is presented in Equation 8.15.

$$rbf^{(h)}_{i,k,q}(t) = C_{i,k} + I^{(h)}_{i,k,q} + B^{(h)}_{i,k,q} \tag{8.15}$$

The interference of higher or equal priority messages is calculated according to Equation 8.16. Similar to the case of response-time analysis for tasks, the messages that are shared among transactions should be excluded by checking the message identifier, i.e., $p \neq k$.

$$I^{(h)}_{i,k,q} = \sum_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\ P_{z,p} \leq P_{i,k} \wedge\ p \neq k \\ \wedge\ l_q \in \mathcal{L}_{z,p} \wedge\ AP(m_{z,p}) = AP(m_{i,k})}} \left\lceil \frac{t}{T_{z,p}} \right\rceil C_{z,p} \tag{8.16}$$

When a message is ready to be forwarded in the switch, it might be blocked by a lower priority message that is already under transmission. The blocking delay is the maximum size of a message among the messages that belong to the same application as $m_{i,k}$ and cross the link l_q.

$$B^{(h)}_{i,k,q} = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge\ P_{z,p} > P_{i,k} \wedge\ p \neq k \\ \wedge\ l_q \in \mathcal{L}_{z,p} \wedge\ AP(m_{z,p}) = AP(m_{i,k})}} \{C_{z,p}\} \tag{8.17}$$

The response time of $m_{i,k}$ crossing the link l_q is computed in Equation 8.18. The period of the message should be decomposed for each link. This can be done in proportion to the load of the links or by equally distributing the load over the links. We refer the reader to the existing works (e.g., [164]) for details about the decomposition of deadlines. Equation 8.18 should be evaluated for all t until the decomposed deadline for link l_q.

$$RT_{i,k,q} = min(t > 0) : sbf^{(h)}_{i,k,q}(t) \geq rbf^{(h)}_{i,k,q}(t) \tag{8.18}$$

The response time of $m_{i,k}$ is derived in Equation 8.19.

$$RT_{i,k} = \sum_{q=1}^{|\mathcal{L}_{i,k}|} RT_{i,k,q} + |\mathcal{L}_{i,k}| \times \epsilon \tag{8.19}$$

8.2.3 Timing analysis of transactions

The end-to-end response time for a transaction is the sum of the response times of tasks and messages that belong to it. The transaction is schedulable if its end-to-end response time, denoted by $RT^{(h)}_i$, is less than or equal to its end-to-end deadline, denoted by $D^{(h)}_i$. The end-to-end response time of transaction $\Gamma^{(h)}_i$ is computed as follows.

$$RT^{(h)}_i = \sum_{\forall \tau_{j,i,k} \in \Gamma^{(h)}_i} RT_{j,i,k} + \sum_{\forall m_{i,k} \in \Gamma^{(h)}_i} RT_{i,k} \tag{8.20}$$

When the tasks and messages in a transaction have independent activation sources, the data age and reaction delays should be computed. In order to calculate these delays, all *reachable* time paths for a transaction should be derived. A set of Boolean functions are presented by Feiertag *et al.* [20] to derive such reachable time paths. However, time paths presented by the authors are only applicable for networks in which the messages cannot be independently initiated. In the HaRTES architecture a message is triggered independent of the sender task. Therefore, independent activations of the messages in the transaction should be included in the time paths. Figure 8.5 shows a transaction with two tasks and one message. The message is triggered by the switch regardless of the sender task (τ_1). In this figure, five time paths (named from A to E) are illustrated. However, not all of these time paths are reachable. For example, time path D is not reachable as the data cannot travel back in time.

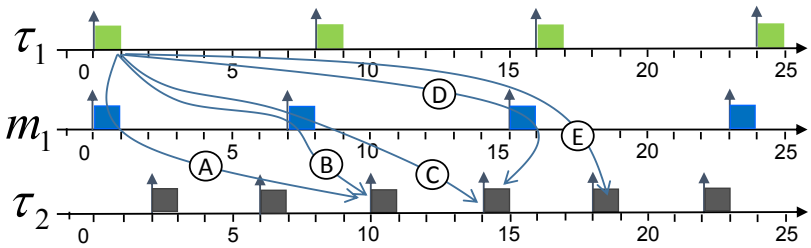

Figure 8.5: An example of time paths including a message.

The algorithm to compute the delays for $\Gamma_i^{(h)}$ is presented in Algorithm 7. In order to compute the longest time path, all reachable time paths are extracted based on the activation patterns of the tasks and messages in the transaction. The extraction is done by the two functions in lines 1 and 2. Basically, every valid time path is verified against the reachability condition. This condition for two neighboring tasks (or task and a message) is as follows. Let $\beta_w(i)$ and $\beta_r(i)$ denote the activation time of the ith instances of the reader and writer tasks respectively. Also, let $RT_w(i)$ denotes the response time of the writer task.

1. The activation time of the writer should be earlier than the reader, i.e., $\beta_r(i) \geq \beta_w(i)$. For instance, time path D in Figure 8.5 does not satisfy this condition between m_1 and τ_2.

2. The execution of the writer and reader should not overlap, i.e., $\beta_r(i) \geq \beta_w(i) + RT_w(i)$. For example, time path A in Figure 8.5 does not satisfy this condition between τ_1 and m_1.

3. The writer and reader can have overlap only if both of them execute on the same node or link, and the priority of the reader is lower than the priority of the writer.

4. There could be a time path in which the output of an instance of the writer is over-written by its next instance. For example, time path E in which the second instance of τ_1 over-writes the data from its previous instance. Such time paths are excluded from the list of reachable time paths.

In order to compute the reaction delay, a subset of reachable time paths is extracted. In this subset, no time path exists that shares the same start instance

of the first task in the transaction and has an earlier end instance of the last task in the transaction.

Algorithm 7 Find data age and reaction delay for transaction $\Gamma_i^{(h)}$

1: TP_Age = $FindAllValidAgeTimePaths(i)$
2: TP_Reac = $FindAllValidReacTimePaths(i)$
3: $Age\_Delay_i^{(h)} = 0; Reac\_Delay_i^{(h)} = 0$
4: **for all** TP_Age **do**
5: New_Age = $ComputeAge(TP\_Age)$
6: **if** $New\_Age > Age\_Delay_i^{(h)}$ **then**
7: $Age\_Delay_i^{(h)} = New\_Age$
8: **end if**
9: **end for**
10: **for all** TP_Reac **do**
11: New_Reac = $ComputeReac(TP\_Reac)$
12: **if** $New\_Reac > Reac\_Delay_i^{(h)}$ **then**
13: $Reac\_Delay_i^{(h)} = New\_Reac$
14: **end if**
15: **end for**
16: **return** $Age\_Delay_i^{(h)}, Reac\_Delay_i^{(h)}$

The function in line 5 of the algorithm provides the age delay of the transaction for a given time path. The function uses Equation 8.21 to compute the age delay [20], where β_n and β_1 represent the activation times of the last task and the first task in the transaction. Note that a transaction cannot start or finish with a message.

$$New\_Age = \beta_n(TP) + RT_{j,i,k} - \beta_1(TP) \tag{8.21}$$

The reaction delay derived in line 11 of the algorithm for a given time path is computed using Equation 8.22, where $Pred(TP)$ is the instance of the first task in the transaction that belongs to the previous reachable time path.

$$New\_Reac = \beta_n(TP) + RT_{j,i,k} - \beta_1(Pred(TP)) \tag{8.22}$$

In Equation 8.21 and Equation 8.22, $RT_{j,i,k}$ represents the response time of the last task in $\Gamma_i^{(h)}$ which executes in node j. The transaction meets its constraints if:

$$Age\_Delay_i^{(h)} \leq Age_i^{(h)} \tag{8.23}$$

$$Reac\_Delay_i^{(h)} \leq Reac_i^{(h)} \tag{8.24}$$

8.2.4 Schedulability of applications

In order to verify the schedulability of applications, the response time of servers corresponding to each application should be evaluated on both nodes and network links. The servers are schedulable if their response times are less than or equal to their respective periods. The response time of a server for $\mathcal{A}^{(h)}$ in a node corresponds to resource r_q. It is recursively calculated using Equation 8.25.

$$RT_q^{(h)} = \sum_{\forall \mathcal{A}^{(f)} \in \mathcal{S} \wedge \Psi_{rq}^{(f)} \leq \Psi_{rq}^{(h)}} \left\lceil \frac{RT_q^{(h)}}{\Pi_{rq}^{(f)}} \right\rceil \Theta_{rq}^{(f)} \tag{8.25}$$

On the network links, the servers use a portion of the bandwidth, i.e., a percentage of the Synchronous or Asynchronous Windows. Therefore, the analysis based on sbf and rbf is used. The Synchronous Window becomes available every EC at a known point in time, which is always after the Guard Window. Figure 8.6 shows the availability of windows and the sbf for the Synchronous Window in link l_q, where L_{EC} is the size of the EC; and LS_q and LA_q are the sizes of the Synchronous and Asynchronous Windows in link l_q respectively. In order to calculate the supply for a given time interval t, we consider that the interval starts at the beginning of the EC as the resource is periodic in a fixed position within the EC. There are three scenarios for the time interval: (a) it finishes before the window, (b) it finishes within the window, and (c) it finishes before the end of the EC but consumes the whole window.

The time interval t in Figure 8.6 is calculated using Equation 8.26, where ρ is a part of the supply in the last EC and Id_q is the idle time. The calculations for the idle time in the servers will be explained later in this section.

$$sbf_q^{(h)}(t) = \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times (LS_q - Id_q) + \rho \tag{8.26}$$

We compute the supply in the last EC for the scenarios (b) and (c) as depicted in Figure 8.6. If the interval finishes in the middle of the Synchronous

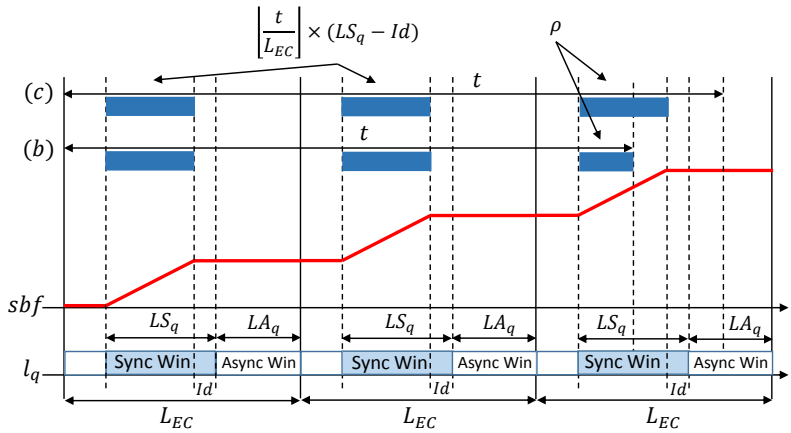

Figure 8.6: Supply bound function for the Synchronous Window.

Window (i.e., scenario (b)), ρ is computed using Equation 8.27. Note that this scenario also covers scenario (a) by using the *max* operation in Equation 8.27.

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LS_q - LA_q), 0\}, \\ &\text{if } \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} \leq t \leq \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - LA_q - Id_q) \end{aligned} \tag{8.27}$$

However, if the time interval finishes after the Synchronous Window but before the end of the EC (i.e., scenario (c)), the calculations for ρ are different as shown in Equation 8.28.

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LS_q + Id_q), 0\}, \\ &\text{if } t > \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - LA_q - Id_q) \end{aligned} \tag{8.28}$$

The Asynchronous Window becomes periodically available at a known point in time within the EC. Hence, the sbf for it is computed in a similar fashion as follows.

$$sbf_q^{(h)}(t) = \left\lfloor \frac{t}{L_{EC}} \right\rfloor \times (LA_q - Id_q) + \rho \tag{8.29}$$

The resource in the last EC (ρ) is computed for the scenarios using Equation 8.30 and Equation 8.31.

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LA_q), 0\}, \\ &\text{if } \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} \leq t \leq \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - Id_q) \end{aligned} \tag{8.30}$$

$$\begin{aligned} &\rho = max\{t - \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} - (L_{EC} - LA_q + Id_q), 0\}, \\ &\text{if } t > \left\lfloor \tfrac{t}{L_{EC}} \right\rfloor \times L_{EC} + (L_{EC} - Id_q) \end{aligned} \tag{8.31}$$

In order to find the idle time, we derive the maximum message size that crosses the window in link l_q and belongs to $\mathcal{A}^{(h)}$. Equation 8.32 calculates the idle time, where ω represents the periodic and sporadic activation pattern if the idle time is calculated in the Synchronous and Asynchronous Windows respectively.

$$Id_q = \max_{\substack{\forall m_{z,p} \in \mathcal{A}^{(h)} \wedge \ l_q \in \mathcal{L}_{z,p} \\ \wedge \ AP(m_{z,p}) = \omega}} \{C_{z,p}\} \tag{8.32}$$

The rbf for the servers within the transmission windows is calculated with Equation 8.33, where the interference from higher priority servers is accounted for.

$$rbf_q^{(h)}(t) = \sum_{\forall \mathcal{A}^{(f)} \in \mathcal{S} \wedge \ \Psi_{rq}^{(f)} \leq \Psi_{rq}^{(h)}} \left\lceil \frac{t}{\Pi_{rq}^{(f)}} \right\rceil \Theta_{rq}^{(f)} \tag{8.33}$$

Finally, the response time of the server is calculated when the sbf is equal to or larger than the rbf, as shown in Equation 8.34.

$$RT_q^{(h)} = min(t > 0) : sbf_q^{(h)} \geq rbf_q^{(h)} \tag{8.34}$$

The system $\mathcal{S}$ is schedulable if all the servers in the nodes and network links are schedulable.

8.3 Case study

In order to show the applicability of the model we present a case study from the vehicular domain. We consider an Autonomous Steering Control (ASC) system, that provides electronic steer control to a vehicle using mechanical and

electronic components. The ASC system uses an Ethernet network for communication as the backbone network in the vehicle. The architecture of the ASC system is depicted in Figure 8.7. The ASC system is distributed over 6 Electronic Control Units (ECUs) and two HaRTES switches. There are four Wheel Control (WC) ECUs; one Steer Control (SC) ECU; and one Collision Avoidance Control (CAC) ECU. Moreover, there is a camera (CAM) that is mounted in front of the vehicle to gather the video frames for collision avoidance purpose.

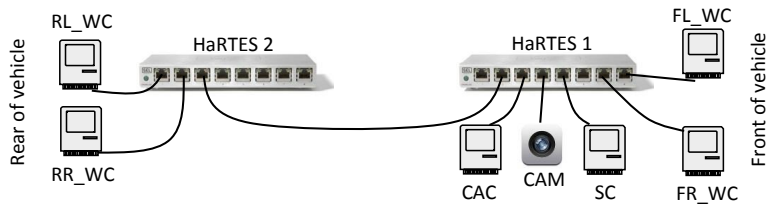

Figure 8.7: The network architecture for the ASC system.

The ASC system is divided into two applications: (i) a conventional steering control, known as Steer-By-Wire (SBW) application, and (ii) the Collision Avoidance (CA) application. Each application together with its transactions is described in the next subsection. It should be noted that the parameters for both applications are inspired by real case systems.

8.3.1 Steer-by-wire application

Each WC ECU gathers the wheel information including the angle and torque of the wheel. This information is sent to the SC ECU via an Ethernet message. Apart from this message, the SC ECU receives several sensor values including steering angle, steering torque and vehicle speed. Based on the information given by these messages, the SC ECU provides feedback on steering torque to the feedback torque actuator. This actuator provides the feeling of turning effect for the driver. Moreover, the SC ECU sends an Ethernet message to each WC ECU. This message includes the steer angle and torque signal which are used by the WC ECUs to control the wheel actuators. Here, we only show the transactions for one wheel, i.e., rear-left (RL) wheel. Two transactions from the RL_WC ECU to the SC ECU are depicted in Figure 8.8. Note that the transactions share the RL_WC_Controller_Task (in the RL_WC ECU), RL_WC_Message (in the network) and SC_Steer_Control_Task (in the SC_ECU).

The notation below each task and message shows its activation pattern, e.g., (I, P, 40ms) shows that the task is periodically activated by an independent source with a period of 40ms.

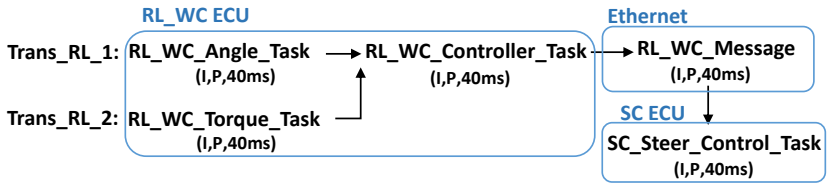

Figure 8.8: Transactions from the rear-left WC ECU to the SC ECU.

In addition, there are three transactions to send information from the SC ECU to WC ECU. These transactions also share tasks and messages which are illustrated in Figure 8.9.

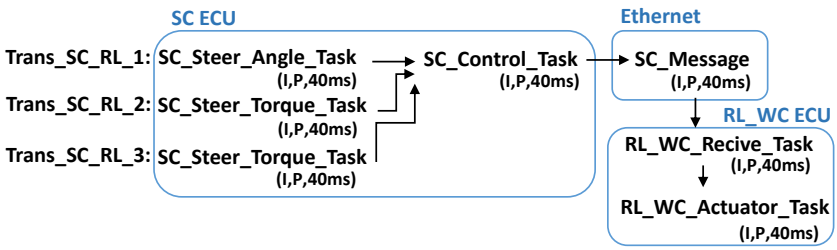

Figure 8.9: Transactions from the SC ECU to the rear-left WC ECU.

Therefore, we have five transactions for each wheel, that makes it 20 transactions in total for the SBW application. In the above transactions, the WCETs of the tasks are as follows: 100 μs for the sensor tasks; 200 μs for the message receiver tasks; and 400 μs for the control tasks. The size of the Ethernet frame is 64 bytes which includes 46 bytes of payload (data). The constraints specified on the transactions include the data age constraint (100 ms) and data reaction constraint (140 ms).

8.3.2 Collision avoidance application

The main function of this application is to detect obstacles by receiving information from a radar and video frames from a camera, and calculates a new steering angle accordingly. This information is collected in the CAC ECU. The CAC ECU sends this information in an Ethernet message to the SC ECU which is responsible for computing the steer actuator signals accordingly. The transactions belonging to this application are illustrated in Figure 8.10. The WCETs of the tasks are selected in a similar way as in the steer-by-wire application. The message size for the control and radar signals is 64 bytes. Whereas, the message size for the video frame is 1500 bytes. For these transactions, the data age and reaction constraints are 110 ms and 150 ms, respectively.

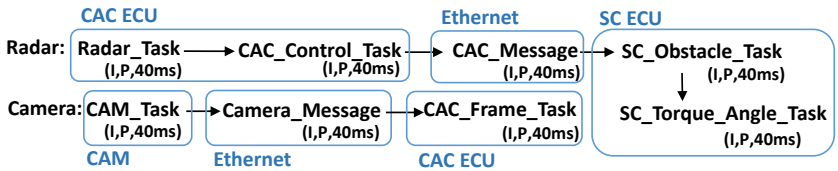

Figure 8.10: Transactions in the CA application.

8.3.3 Resource reservation

The two applications share the SC ECU and network link from the HaRTES 1 switch to the SC ECU. The period and budget of the servers in all the nodes and links that are not shared between the two applications are 20 ms and 19 ms respectively. The period and budget of the server in the SC ECU for the SBW application are equal to 20 ms and 13 ms respectively. Whereas, the period and budget of the server in the SC ECU for the CA application are equal to 20 ms and 6 ms respectively. The priority of the server for the SBW application is higher than the priority of the server for the CA application. The servers are defined for the network link between the HaRTES 1 switch and the SC ECU in a similar way with the same attributes.

8.3.4 Results of the case study

The end-to-end response time, data age and data reaction delays of all the transactions are computed using the presented analysis. The analysis results for all

the transactions are presented in Table 8.1. All timing parameters are in milliseconds. By comparing the computed end-to-end response times and delays with the specified end-to-end constraints, we see that all the transactions are schedulable.

Transactions	E2E RT	Age Delay	Reaction Delay
Trans_RL_1	24.36	94.1	134.1
Trans_RL_2	24.36	94.1	134.1
Trans_SC_RL_1	38.72	82.1	122.1
Trans_SC_RL_2	39.52	83.5	123.5
Trans_SC_RL_3	38.72	82.1	122.1
Radar	60.68	106.1	146.1
Camera	32.50	106.1	146.1

Table 8.1: Case study results.

8.4 Conclusions

This chapter extends the results from the multi-hop communication to provide an end-to-end framework for use in distributed embedded systems. A model with reservation of resources in the distributed embedded systems is proposed where transactions can have several tasks in one node. The model provides possibility of reserving resources for a particular application that is distributed over several nodes via a network. A timing analysis including an end-to-end response time and end-to-end delays is presented. An automotive case study showed the applicability of the model in an industrial application.

Chapter 9

Conclusions and Future Work

The tendency to use advanced embedded equipment and sophisticated software in distributed embedded systems is increasing. Therefore, the requirements on the functionality and performance of network communication are becoming overly intricate. These requirements include real-time guarantees, run-time adaptation ability, efficient resource utilization and high flexibility during the development of the system. This dissertation focused on addressing such requirements in two Ethernet-based network architectures, namely the FTT-SE architecture and the HaRTES architecture. The architectures are based on the master-slave technique with one master module controlling the traffic transmission through the switch. Moreover, they support time-triggered and event-triggered traffic types by reserving resources for each of which. This reservation mechanism allows the network to transmit different types of traffic with temporal isolation. Both architectures have an admission control and a QoS management that make them capable of performing run-time adaptation. Considering the abilities of the mentioned architectures, they have emerged as promising solutions that can cater for the mentioned requirements. However, the FTT-SE and HaRTES architectures were originally developed for single-hop networks. This dissertation presented solutions to provide a multi-hop communication as a response to demands from industrial applications with hundreds of nodes. The presented solutions maintained the prominent features of the architectures including the timeliness, run-time adaptation property and resource utilization.

9.1 Summary of the dissertation

The thesis statement presented in Chapter 1 emphasized three aspects. These three aspects include: (i) providing a multi-hop communication without jeopardizing real-time properties, (ii) enabling end-to-end adaptable real-time communication, and (iii) providing resource reservation in distributed embedded systems. Each of these aspects opened a research question, which are reviewed again in this section.

9.1.1 Research question 1 (RQ1)

RQ1: Which solutions offer better performance in terms of resource usage to extend the FTT-SE and HaRTES architectures for multi-hop communication?

This research question addressed the first aspect of the thesis statement. In the context of the FTT-SE architecture a solution to extend it for multi-hop communication was already proposed, which is called the single-master FTT-SE architecture. This dissertation proposed two additional architectures, namely the multi-master and hybrid architectures. The former architecture uses multiple master nodes, each connected to one switch, to schedule the traffic. The latter architecture reduced the number of master nodes and assigned one master node for a group of switches (which is called a cluster). Response time analysis for both architectures are proposed, considering the behavior of the scheduling algorithms. The architectures are compared against each other and against the single-master architecture with respect to resource utilization. The evaluation is done based on the proposed response time analysis. The results showed that the multi-master architecture uses less resources compared to the single-master architecture with the same traffic settings, mostly in large networks. However, the hybrid architecture performs even better with respect to the resource utilization, in particular for large-scale networks. This success for the hybrid architecture is mainly due to splitting the network into several clusters and dividing the traffic into internal (transmitting within a cluster) and external (transmitting beyond a cluster).

In addition, the dissertation proposed two solutions to provide a multi-hop communication in the context of the HaRTES architecture. The first solution is called Distributed Global Scheduling (DGS) in which each and every switch in the route of traffics involved in the scheduling and forwarding of the traffic. Therefore, the traffic is transmitted hop-by-hop and in each hop the switch

schedules the traffic for the upcoming EC. In contrast, the second solution modified the output queues of the HaRTES switch. Basically, the FIFO queues are replaced with priority-based queues for real-time traffic. Therefore, whenever the traffic is activated it is transmitted via the priority-based queues, albeit within their associated windows. The traffic can be transmitted as long as there is enough time within the window. Therefore, the traffic can traverse through multiple switches in one EC only. This reduces the number of buffering instances of a message when traversing through the switches, hence the method is called Reduced Buffering Scheme (RBS). The dissertation also presented response time analyses for the solutions. Comparisons are performed to show the performance of each solution. The results showed that the RBS scheduling method can deliver the traffic faster compared to when using the DGS scheduling method. This performance advantage is more highlighted for high priority messages due to the use of priority-based queues. The RBS method is implemented in the HaRTES switch in order to validate the method. In general, the hybrid FTT-SE architecture and the RBS method in the HaRTES architecture offer better performance with respect to resource utilization. In order to improve the response time of messages, algorithms to configure the MTU size of messages are proposed in this dissertation, where the results showed that the response time of messages can be improved by around 20%. Chapters 4, 5 and 6 presented the above results.

9.1.2 Research question 2 (RQ2)

RQ2: Which dynamic reconfiguration protocols can be used in the FTT-SE and HaRTES networks that are compatible with their traffic scheduling and forwarding algorithms?

The second research question addressed the second aspect of the thesis statement, which is enabling run-time adaptation in multi-hop communication. For the multi-hop FTT-SE architecture, two reconfiguration protocols are defined in Chapter 7. The first protocol involves all master nodes to the reconfiguration process in the sense that all master nodes verify a change request simultaneously. This protocol requires synchronized master nodes in terms of scheduling, request handling and applying the changes. This protocol is called distributed reconfiguration. The second protocol utilizes one master node to take care of all requests in the network, that designates the centralized reconfiguration. A comparison is done between the protocols and the results showed

that the centralized reconfiguration protocol requires less complexity in implementation and less signaling overhead.

For the multi-hop HaRTES architecture, the RBS scheduling method is chosen due to its success compared with the DGS method. This dissertation presented a reconfiguration protocol that is compatible with the RBS method, which is called the clustered reconfiguration protocol. The network is divided into several clusters and one switch in each cluster takes the responsibility of verifying change requests. Moreover, a particular switch in the entire network takes the responsibility of coordinating clusters. Evaluations of the clustered reconfiguration showed that the whole process is bounded in time, allowing for hard real-time requirements.

In addition, the work proved the correctness of both the centralized and the clustered reconfiguration protocol using the UPPAAL tool, that is a model checker. For this purpose models of the protocols are defined in the tool and some properties are verified. The tool provided evidences that the proposed protocols are deadlock-free and that they always provide results with either acceptance or rejection of a request.

9.1.3 Research question 3 (RQ3)

RQ3: How can we extend the results from the multi-hop communication to obtain an end-to-end resource reservation framework, including computational and communication resources?

Considering the third aspect of the thesis statement, the target is distributed embedded systems. Therefore, Chapter 8 aimed at extending the results obtained in the context of multi-hop communication to an end-to-end framework that includes computational nodes. To do so, the dissertation presented a new end-to-end resource reservation framework that supports reservation on both communication and computation resources. A general transactional model is considered such that transactions can contain several tasks in each node and several messages in the network. Moreover, tasks and messages can have different activation patterns. Various transactions that share resources are grouped as one application. Applications in this framework can be designed and implemented separately of how and with what they will be integrated with later on. Then, in the final phase of the development, they can be integrated. This brings flexibility during the development phase. The dissertation also presented an end-to-end timing analysis that supports end-to-end reservation. The usabil-

ity of the proposed framework is verified with the help of an automotive case study.

9.2 Future work

There are still many remaining issues that need to be addressed. This section provides a number of interesting issues stemming from our current contributions.

9.2.1 Multi-hop communication

In general, the proposed solutions to support multi-hop communication over the FTT-SE and HaRTES architectures are not fully implemented. Although the RBS scheduling method is already implemented in the HaRTES switch, the validation and experiments have so far only been done on a small setting with three nodes. Therefore, the main future work is to develop the presented solutions on hardware and experimentally validate them. With respect to the response time analysis, there is still some pessimism in the response time analyses of all presented solutions. This pessimism is discussed for each analysis and some sources of pessimism are already identified. However, the solutions to remove this pessimism have not been fully investigated yet, which is part of the future research direction. In particular, the release offset is not presented for the traffic which can help in the improvement of the message transmission time, and it has not been included in the response time analysis. To achieve a consistency in scheduling and transmission of traffic within the ECs, the master nodes in the FTT-SE architecture and switches in the HaRTES architecture should be timely synchronized. In this dissertation we proposed to use signaling mechanisms to provide such a synchronization. This may introduce jitter between the ECs in each node and switches. Therefore, it may be worth to investigate an implementation of a clock synchronization mechanism for the architectures. A preliminary study has been done [154] in the context of the FTT-SE architecture. However, deep study and investigating the effects of clock synchronization on the performance of the network should be done. In addition, the performance of the multi-hop FTT-SE architecture compared to the multi-hop HaRTES architecture is not fully investigated. This direction of future work is also necessary to draw the benefits of each architecture in different applications.

9.2.2 Dynamic reconfiguration

The proposed reconfiguration protocols have not been implemented yet. With such an implementation, the protocols can be assessed experimentally, in particular against the timing behavior of the message transmission. Moreover, the protocols lack a study on impact of message losses for the sake of robustness. Although this issue is mainly relevant in the area of fault tolerant systems, providing a detection and recovery solution for message losses is interesting. Another issue regarding the proposed protocols is that the reconfiguration is software-initiated. This means that a node issues a request for a change in the traffic setting. It could be interesting to lift the protocols for other types of reconfiguration such as removing or inserting a new device in a network during run-time. Regarding the model-checking there is still some work left that can be done. For instance, temporal features can be verified. These properties can be verified using the UPPAAL tool, including the time needed for the reconfiguration process.

9.2.3 End-to-end reservation

The presented end-to-end framework requires that the user specifies the reservation parameters such as server period, budget and priority. The server design plays a critical role in the end-to-end timing analysis and schedulability of the applications. Therefore, the main future work aims at providing guidelines to the user to assign these parameters efficiently. This requires the definition of an optimization problem along with usage of efficient and linear-complexity algorithms to find the best parameters. Another step for a future research direction is to use the end-to-end framework in the development of software architectures for industrial applications. A preliminary work has been done in this area [165] where the Rubus Component Model (RCM) [166] is extended to fulfill the requirements of modeling such a framework in the vehicular embedded system domain.

Bibliography

[1] F. Gomez-Molinero. Real-time requirement of media control applications. In *19th Euromicro Conference on Real-Time Systems*, July 2007.

[2] C.-S. Cho, B.-M. Chung, and M.-J. Park. Development of real-time vision-based fabric inspection system. *IEEE Transactions on Industrial Electronics*, 52(4):1073–1079, August 2005.

[3] C.-L. Hwang and C.-Y. Shih. 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, March 2009.

[4] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch. Time-Triggered Ethernet: TTEthernet. In *Time-Triggered Communication, R. Obermaisser, editor, CRC Press*, 2011.

[5] Z. Hanzalek, P. Burget, and P. Sucha. Profinet IO IRT message scheduling. In *21st Euromicro Conference on Real-Time Systems*, July 2009.

[6] Ethernet POWERLINK Standardisation Group. *EPSG Draft Standard 301 Ethernet POWERLINK Communication Profile Specification Version 1.2.0*, 2013.

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

[8] G. Carvajal, M. Figueroa, R. Trausmuth, and S. Fischmeister. 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*, April 2013.

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

[10] IEEE. 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. Technical report, IEEE, 2011.

[11] IEEE. IEEE Std. 802.1qat, IEEE standard for local and metropolitan area networks, virtual bridged local area networks, amendment 14: Stream Reservation Protocol. Technical report, IEEE, 2011.

[12] IEEE. 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. Technical report, IEEE, 2011.

[13] G. Alderisi, G. Patti, and L.L. Bello. Introducing support for scheduled traffic over IEEE audio video bridging networks. In *18th IEEE Conference on Emerging Technologies Factory Automation*, September 2013.

[14] Time-sensitive networking task group, available at http://www.ieee802.org/1/pages/tsn.html, access date: May 2016.

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

[16] R. Santos, A. Vieira, P. Pedreiras, A. Oliveira, L. Almeida, R. Marau, and T. Nolte. Flexible, efficient and robust real-time communication with server-based ethernet switching. In *8th IEEE International Workshop on Factory Communication Systems*, May 2010.

[17] P. Pedreiras and L. Almeida. The flexible time-triggered (FTT) paradigm: an approach to QoS management in distributed real-time systems. In *The International Parallel and Distributed Processing Symposium*, April 2003.

[18] F. Yekeh, M. Pordel, L. Almeida, M. Behnam, and P. Portugal. Exploring alternatives to scale FTT-SE to large networks. In *6th IEEE International Symposium on Industrial Embedded Systems*, June 2011.

[19] R. Marau, M. Behnam, Z. Iqbal, P. Silva, L. Almeida, and P. Portugal. Controlling multi-switch networks for prompt reconfiguration. In *9th International Workshop on Factory Communication Systems*, May 2012.

[20] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, December 2008.

[21] R. Snieder and K. Larner. *The Art of Being a Scientist: A Guide for Graduate Students and their Mentors*. Cambridge University Press, 2009.

[22] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte. Performance analysis of master-slave multi-hop switched ethernet networks. In *8th IEEE International Symposium on Industrial Embedded Systems*, June 2013.

[23] M. Ashjaei, P. Pedreiras, M. Behnam, R. J. Bril, L. Almeida, and T. Nolte. Response time analysis of multi-hop HaRTES ethernet switch networks. In *9th International Workshop on Factory Communication Systems*, May 2014.

[24] M. Ashjaei, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte. Reduced buffering solution for multi-hop HaRTES switched Ethernet networks. In *20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications*, August 2014.

[25] M. Ashjaei, L. Silva, M. Behnam, P. Pedreiras, R. J. Bril, L. Almeida, and T. Nolte. Improved message forwarding for multi-hop HaRTES real-time Ethernet networks. *Journal of Signal Processing Systems*, May 2015.

[26] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte. MTU configuration for real-time switched Ethernet networks. *Journal of Systems Architecture*, April 2016.

[27] M. Ashjaei, P. Pedreiras, M. Behnam, L. Almeida, and T. Nolte. Dynamic reconfiguration in multi-hop switched Ethernet networks. In *6th Workshop on Adaptive and Reconfigurable Embedded Systems*, April 2014.

[28] M. Ashjaei, Y. Du, L. Almeida, M. Behnam, and T. Nolte. Dynamic reconfiguration in HaRTES switched Ethernet networks. In *12th IEEE World Conference on Factory Communication Systems*, May 2016.

[29] M. Ashjaei, S. Mubeen, M. Behnam, L. Almeida, and T. Nolte. End-to-end resource reservations in distributed embedded systems. In *22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2016.

[30] M. Barr and A. Massa. Programming embedded systems, 2nd edition. In *O'Reilly Media*, October 2006.

[31] R. N. Charette. This car runs on code. In *Spectrum, IEEE, 46(2)*, February 2009.

[32] G. C. Buttazzo. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications*. Springer-Verlag TELOS, 2004.

[33] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8(2):173–198, March/May 1995.

[34] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Systems*, 28(2):101–155, February 2004.

[35] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, 1973.

[36] I. Stoica, H. Abdel-Wahab, K. Jeffay, S. K. Baruah, J. E. Gehrke, and C. G. Plaxton. A proportional share resource allocation algorithm for real-time, time-shared systems. In *17th IEEE Real-Time Systems Symposium*, December 1996.

[37] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic task scheduling for hard-real-time systems. *Real-Time Systems Journal*, 1(1):27–60, June 1989.

[38] J. K. Strosnider, J. P. Lehoczky, and Lui Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, January 1995.

[39] M. Spuri and G. C. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *Real-Time Systems Symposium*, December 1994.

[40] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems*, 10:179–210, 1996.

[41] L. Abeni and G. Buttazzo. Resource reservation in dynamic real-time systems. *Real-Time System*, 27(2):123–167, July 2004.

[42] S. Saewong, R. Rajkumar, J. P. Lehoczky, and M. H. Klein. Analysis of hierarchical fixed-priority scheduling. In *14th Euromicro Conference on Real-Time Systems*, July 2002.

[43] G. Lipari and E. Bini. Resource partitioning among real-time applications. In *15th Euromicro Conference on Real-Time Systems*, July 2003.

[44] Z. Deng and J. W. S. Liu. Scheduling real-time applications in an open environment. In *18th IEEE Real-Time Systems Symposium*, December 1997.

[45] Z. Deng, J. W. S. Liu, and J. Sun. A scheme for scheduling hard real-time applications in open system environment. In *9th Workshop on Real-Time Systems*, June 1997.

[46] G. Lipari and S. Baruah. A hierarchical extension to the constant bandwidth server framework. In *7th IEEE Real-Time Technology and Applications Symposium*, May 2001.

[47] M. Broy, IH. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2):356–373, February 2007.

[48] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1st edition, 1997.

[49] *The FlexRay Communication System Specification, version 2.1*. 2005.

[50] L. Almeida, P. Pedreiras, and J.AG. Fonseca. The FTT-CAN protocol: why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, December 2002.

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

[52] Robert Bosch GmbH. CAN Specification Version 2.0. Postfach 30 02 40, D-70442 Stuttgart, 1991.

[53] ISO 11898-1. Road Vehicles interchange of digital information controller area network (CAN) for high-speed communication, ISO Standard-11898, Nov. 1993.

[54] L. Sha and S.S. Sathaye. A systematic approach to designing distributed real-time systems. *IEEE Computer*, 26(9):68–78, September 1993.

[55] G. Buttazzo. *Hard Real-Time Computing Systems, third edition*. Springer, 2011.

[56] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal*, 8(5):284–292, September 1993.

[57] C.L. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, January 1973.

[58] M. S. Fineberg and O. Serlin. Multiprogramming for hybrid computation. In *Proceedings of the Fall Joint Computer Conference*, pages 1–13, 1967.

[59] M. Joseph and P. Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, March 1986.

[60] M. Gonzalez Harbour and J.C. Palencia. Response time analysis for tasks scheduled under EDF within fixed priorities. In *24th IEEE Real-Time Systems Symposium*, pages 200–209, December 2003.

[61] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.

[62] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng. Schedulability analysis of Ethernet AVB switches. In *20th IEEE International Conference on embedded and Real-Time Computing Systems and Applications*, August 2014.

[63] R. Santos, M. Behnam, T. Nolte, P. Pedreiras, and L. Almeida. Multi-level hierarchical scheduling in Ethernet switches. In *International Conference on Embedded Software*, October 2011.

[64] J. Leboudec and P. Thiran. *Network Calculus*. Berlin, Germany: Spriger-Verlag, 2001.

[65] A. Mifdaoui, F. Frances, and C. Fraboul. Performance analysis of a master/slave switched Ethernet for military embedded applications. *IEEE Transactions on Industrial Informatics*, 6(4):534–547, November 2010.

[66] M. Zhang, J. Shi, T. Zhang, and Y. Hu. Hard real-time communication over multi-hop switched Ethernet. In *The IEEE International Conference on Networking, Architecture, and Storage*, June 2008.

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

[68] R. Queck. Analysis of Ethernet AVB for automotive networks using network calculus. In *IEEE International Conference on Vehicular Electronics and Safety*, July 2012.

[69] M. Manderscheid and F. Langer. Network calculus for the validation of automotive Ethernet in-vehicle network configurations. In *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, October 2011.

[70] L. Lenzini, L. Martorini, E. Mingozzi, and G. Stea. Tight end-to-end per-flow delay bounds in FIFO multiplexing sink-tree networks. *Elsevier Performance Evaluation*, 63(9):956987, October 2006.

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

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

[73] S. Martin and P. Minet. Schedulability analysis of flows scheduled with FIFO: application to the expedited forwarding class. In *20th International Parallel and Distributed Processing Symposium*, April 2006.

[74] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th Real-Time Systems Symposium*, December 1990.

[75] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In *14th IEEE Conference on Emerging Technologies Factory Automation*, September 2009.

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

[77] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Applying trajectory approach with static priority queuing for improving the use of available AFDX resources. *Real-Time Systems Journal*, 48(1):101–133, 2012.

[78] G. Kemayo, F. Ridouard, H. Bauer, and P. Richard. Optimistic problems in the trajectory approach in FIFO context. In *18th IEEE Conference on Emerging Technologies Factory Automation*, September 2013.

[79] X. Li, O. Cros, and L. George. The trajectory approach for AFDX FIFO networks revisited and corrected. In *20th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2014.

[80] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Worst-case backlog evaluation of avionics switched ethernet networks with the trajectory approach. In *24th Euromicro Conference on Real-Time Systems*, July 2012.

[81] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for end-to-end response-time and delay analysis in the industrial tool suite: Issues, experiences and a case study. *Computer Science and Information Systems*, 10(1), 2013.

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

[83] D. TamasSelicean, P. Pop, and W. Steiner. Timing analysis of rate constrained traffic for the TTEthernet communication protocol. In *18th IEEE International Symposium on Real-Time Distributed Computing*, April 2015.

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

[85] W. Steiner. An evaluation of SMT-Based schedule synthesis for Time-Triggered multi-hop networks. In *31st IEEE Real-Time Systems Symposium*, November 2010.

[86] S. S. Craciunas, R. S. Oliver, and V. Ecker. Optimal static scheduling of real-time tasks on distributed time-triggered networked systems. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation*, September 2014.

[87] P. G. Peon, H. Kopetz, and W. Steiner. Towards a reliable and high-speed wireless complement to TTEthernet. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation*, September 2014.

[88] E. Lisova, E. Uhlemann, J. kerberg, and M. Bjrkman. Towards secure wireless TTEthernet for industrial process automation applications. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation*, September 2014.

[89] W. Steiner, F. Bonomi, and H. Kopetz. Towards synchronous deterministic channels for the internet of things. In *IEEE World Forum on Internet of Things*, March 2014.

[90] L. Lo Bello. The case for Ethernet in automotive communications. *SIGBED Review*, 8(4, pp. 7-15), December 2011.

[91] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin. Intra-vehicle networks: A review. *IEEE Transaction on Intelligent Transportation Systems*, 16(2), April 2015.

[92] T. Steinbach, Hyung-Taek Lim, F. Korf, T.C. Schmidt, D. Herrscher, and A. Wolisz. Tomorrow's in-car interconnect? a competitive evaluation of IEEE 802.1 AVB and time-triggered ethernet (as6802). In *Vehicular Technology Conference*, September 2012.

[93] G. Alderisi, A. Caltabiano, G. Vasta, G. Iannizzotto, T. Steinbach, and L. Lo Bello. Simulative assessments of IEEE 802.1 Ethernet AVB and time-triggered ethernet for advanced driver assistance systems and in-car infotainment. In *Vehicular Networking Conference*, November 2012.

[94] G. Alderisi, G. Iannizzotto, and L. Lo Bello. Towards 802.1 Ethernet AVB for advanced driver assistance systems: a preliminary assessment. In *17th IEEE Conference on Emerging Technologies Factory Automation*, September 2012.

[95] J. Imtiaz, J. Jasperneite, and S. Schriegel. A proposal to integrate process data communication to IEEE 802.1 Audio Video Bridging (AVB). In *16th IEEE Conference on Emerging Technologies Factory Automation*, September 2011.

[96] J. Jasperneite, J. Imtiaz, M. Schumacher, and K. Weber. A proposal for a generic real-time ethernet system. *IEEE Transactions on Industrial Informatics*, 5(2), May 2009.

[97] S. Thangamuthu, N. Concer, P.J.L.Cuijpers, and J.J. Lukkien. Analysis of ethernet-switch traffic shapers for in-vehicle networking applications. In *Design, Automation and Test in Europe Conference and Exhibition*, March 2015.

[98] H. Hoang and M. Jonsson. Switched real-time Ethernet in industrial applications - deadline partitioning. In *9th Asia-Pacific Conference on Communications*, September 2003.

[99] S. Varadarajan and T. Chiueh. EtheReal: a host-transparent real-time fast ethernet switch. In *6th International Conference on Network Protocols*, October 1998.

[100] *Profibus International, Application layer protocol for decentralized periphery and distributed automation, specification for PROFINET, IEC 61158-6-IO/FDIS*. October 2007.

[101] *Profibus International, Application layer services for decentralized periphery and distributed automation, specification for PROFINET, IEC 61158-5-IO/FDIS*. October 2007.

[102] G. Prytz. A performance analysis of EtherCAT and PROFINET IRT. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, September 2008.

[103] R. Marau. *Real-time communications over switched Ethernet supporting dynamic QoS management*. PhD Thesis, University of Aveiro, Aveiro, Portugal, 2009.

[104] R. Santos. *Enhanced Ethernet Switching Technology for Adaptive Hard Real-Time Applications*. PhD Thesis, University of Aveiro, Aveiro, Portugal, 2010.

[105] Y. Lee and K. Park. Meeting the real-time constraints with standard Ethernet in an in-vehicle network. In *IEEE Intelligent Vehicles Symposium*, June 2013.

[106] C. A. Kent and J. C. Mogul. Fragmentation considered harmful. In *In ACM SIGCOMM*, pages 390–401, 1987.

[107] E. Creedon and M. Manzke. Impact of fragmentation strategy on ethernet performance. In *International Conference on Network and Parallel Computing*, October 2009.

[108] J. Chen, L. Gong, Y. Yang, and P. Zeng. Average performance of packet network. In *6th International Conference on ITS Telecommunications Proceedings*, June 2006.

[109] Y. Sankarasubramaniam, I.F. Akyildiz, and S.W. McLaughlin. Energy efficiency based packet size optimization in wireless sensor networks. In *International Workshop on Sensor Network Protocols and Applications*, 2003.

[110] S.R. Chudasama and S.D. Trapasiya. Packet size optimization in wireless sensor network using cross-layer design approach. In *International Conference on Advances in Computing, Communications and Informatics*, September 2014.

[111] M.C. Vuran and I.F. Akyildiz. Cross-layer packet size optimization for wireless terrestrial, underwater, and underground sensor networks. In *27th Conference on Computer Communications*, April 2008.

[112] L. T. Jung and A. B. Abdullah. Underwater wireless network energy efficiency and optimal data packet size. In *International Conference on Electrical, Control and Computer Engineering*, June 2011.

[113] C.K. Kodikara, S.T. Worrall, and A.M. Kondoz. Optimal settings of maximum transfer unit (MTU) for efficient wireless video communications. *IEE Proceedings Communications*, 152, October 2005.

[114] Y. Lin and H. Latchman. On the effects of maximum transmission unit in power line communication networks. In *IEEE International Symposium on Power Line Communications and Its Applications*, March 2007.

[115] N. Yaakob and I. Khalil. Packet size optimization for congestion control in pervasive healthcare monitoring. In *10th IEEE International Conference on Information Technology and Applications in Biomedicine*, November 2010.

[116] M. Younis, O. Farrag, and W. D'Amico. Packet size optimization for increased throughput in multi-level security wireless networks. In *Military Communications Conference*, October 2009.

[117] T. Frantti, M. Majanen, and T. Sukuvaara. Delay based packet size control in Wireless Local Area Networks. In *Second International Conference on Ubiquitous and Future Networks*, June 2010.

[118] R. J. Bril, J. J. Lukkien, and W. F. J. Verhaegh. Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption revisited. In *19th Euromicro Conference on Real-Time Systems*, July 2007.

[119] M. Bertogna, O. Xhani, M. Marinoni, F. Esposito, and G. Buttazzo. Optimal selection of preemption points to minimize preemption overhead. In *23rd Euromicro Conference on Real-Time Systems*, July 2011.

[120] M. Bertogna, G. Buttazzo, and Gang Yao. Improving feasibility of fixed priority tasks using non-preemptive regions. In *2011 IEEE 32nd Real-Time Systems Symposium*, November 2011.

[121] M. Behnam, R. Marau, and P. Pedreiras. Analysis and optimization of the MTU in real-time communications over switched Ethernet. In *16th IEEE International Conference on Emerging Technologies Factory Automation*, September 2011.

[122] M. Ashjaei, M. Behnam, L. Almeida, and T. Nolte. MTU assignment in a master-slave switched Ethernet network. In *12th International Workshop on Real-Time Networks*, July 2013.

[123] J. Kramer and J. Magee. Dynamic configuration for distributed systems. *IEEE Transaction in Software Engineering*, April 1985.

[124] J. Purtilo C. Hofmeister, E. White. SURGEON: a packager for dynamically reconfigurable distributed applications. In *International Workshop on Configurable Distributed Systems*, March 1992.

[125] K.W. Tindell, A. Burns, and A.J. Wellings. Mode changes in priority preemptively scheduled systems. In *Real-Time Systems Symposium, 1992*, December 1992.

[126] D. Sciuto M.D. Santambrogio, V. Rana. Operating system support for online partial dynamic reconfiguration management. In *International Conference on Field Programmable Logic and Applications*, September 2008.

[127] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 2004.

[128] P. Kumar, N. Stoimenov, and L. Thiele. An algorithm for online reconfiguration of resource reservations for hard real-time systems. In *24th Euromicro Conference on Real-Time Systems*, July 2012.

[129] M. Garcia Valls, I.R. Lopez, and L.F. Villar. iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems. *IEEE Transactions on Industrial Informatics*, February 2013.

[130] M. Garcia-Valls, P. Basanta-Val, and I. Estevez-Ayres. Real-time reconfiguration in multimedia embedded systems. *IEEE Transactions on Consumer Electronics*, 2011.

[131] U. Brinkschulte, E. Schneider, and F. Picioroaga. Dynamic real-time reconfiguration in distributed systems: timing issues and solutions. In *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, May 2005.

[132] L. Wisniewski, M. Schumacher, J. Jasperneite, and C. Diedrich. Increasing flexibility of time triggered Ethernet based systems by optimal greedy scheduling approach. In *20th Conference on Emerging Technologies Factory Automation*, September 2015.

[133] L. Wisniewski, S. Chahar, and J. Jasperneite. Seamless reconfiguration of time triggered Ethernet based protocols. In *IEEE World Conference on Factory Communication Systems*, May 2015.

[134] R. Marau, L. Almeida, M. Sousa, and P. Pedreiras. A middleware to support dynamic reconfiguration of real-time networks. In *IEEE Conference on Emerging Technologies and Factory Automation*, September 2010.

[135] R. Marau, L. Almeida, M. Sousa, and P. Pedreiras. A middleware to support dynamic reconfiguration of real-time networks. In *15th IEEE Conference on Emerging Technologies and Factory Automation*, September 2010.

[136] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. *ACM symposium on operating systems principles*, October 2003.

[137] Xiang (Alex) Feng. Towards real-time enabled microsoft windows. In *5th ACM International Conference on Embedded Software*, 2005.

[138] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *24th IEEE International Real-Time Systems Symposium*, December 2003.

[139] L. Almeida and P. Pedreiras. Scheduling within temporal partitions: Response-time analysis and server design. In *4th ACM International Conference on Embedded Software*, October 2004.

[140] J. Loeser and H. Haertig. Low-latency hard real-time communication over switched ethernet. In *16th Euromicro Conference on Real-Time Systems*, June 2004.

[141] Z. Iqbal, L. Almeida, R. Marau, M. Behnam, and T. Nolte. Implementing hierarchical scheduling on COTS Ethernet switches using a master/slave approach. In *7th IEEE International Symposium on Industrial Embedded Systems*, June 2012.

[142] M. Sojka, P. Píša, D. Faggioli, T. Cucinotta, F. Checconi, Z. Hanzálek, and G. Lipari. Modular software architecture for flexible reservation mechanisms on heterogeneous resources. *Journal of System Architecture*, April 2011.

[143] G. Dermler, W. Fiederer, I. Barth, and K. Rothermel. A negotiation and resource reservation protocol (NRP) for configurable multimedia applications. In *3rd IEEE International Conference on Multimedia Computing and Systems*, June 1996.

[144] T. Cucinotta and L. Palopoli. QoS control for pipelines of tasks using multiple resources. *IEEE Transactions on Computers*, March 2010.

[145] N. Khalilzad, M. Ashjaei, L. Almeida, M. Behnam, and T. Nolte. Adaptive multi-resource end-to-end reservations for component-based distributed real-time systems. In *13th IEEE Symposium on Embedded Systems For Real-time Multimedia*, October 2015.

[146] K. Lakshmanan and R. Rajkumar. Distributed resource kernels: OS support for end-to-end resource isolation. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, April 2008.

[147] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *18th IEEE Real-Time Systems Symposium*, December 1997.

[148] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky. Integrated resource management and scheduling with multi-resource constraints. In *25th IEEE International Real-Time Systems Symposium*, 2004.

[149] A. B. de Oliveira, A. Azim, S. Fischmeister, R. Marau, and L. Almeida. D-RES: Correct transitive distributed service sharing. In *IEEE Emerging Technology and Factory Automation*, September 2014.

[150] R. Marau, P. Pedreiras, and L. Almeida. Asynchronous traffic signaling over switched Ethernet protocols. In *6th International Workshop on Real Time Networks*, July 2007.

[151] M. Ashjaei, M. Liu, M. Behnam, A. Mifdaoui, L. Almeida, and T. Nolte. Worst-case delay analysis of master-slave switched Ethernet networks. In *2nd International Workshop on Worst-Case Traversal Time*, December 2012.

[152] M. Behnam, T. Nolte, and R.J. Bril. Bounding the number of self-blocking occurrences of SIRAP. In *31th IEEE Real-Time Systems Symposium*, December 2010.

[153] M. Ashjaei, M. Behnam, and T. Nolte. SEtSim: A modular simulation tool for switched Ethernet networks. *Journal of Systems Architecture*, February 2016.

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

[155] H.-T. Lim, K. Weckemann, and D. Herrscher. performance study of an in-car switched Ethernet network without prioritization. In *Third international conference on Communication technologies for vehicles*. Springer-Verlag, 2011.

[156] H.-T. Lim, L. Volker, and D. Herrscher. Challenges in a future IP/Ethernet-based in-car network for real-time applications. In *48th Design Automation Conference*, June 2011.

[157] M. Huang, K. Lim, and J. Cong. A scalable, high-performance customized priority queue. In *24th International Conference on Field Programmable Logic and Applications*, September 2014.

[158] C. Xu and H. Gharavi. A low-complexity tree-search algorithm to decode diversity-oriented block codes with inter-symbol interference. In *IEEE Int. Conference on Communications*, June 2012.

[159] G. Behrmann, R. David, and K. G. Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems*. Springer, 2004.

[160] R. Wang, X. Song, and M. Gu. Modelling and verification of program logic controllers using timed automata. *Software, IET*, 1(4):127–131, 2007.

[161] G. Rodriguez-Navas and J. Proenza. Using timed automata for modeling distributed systems with clocks: Challenges and solutions. *IEEE Transactions on Software Engineering*, 39(6):857–868, june 2013.

[162] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using EDP resource models. In *28th IEEE International Real-Time Systems Symposium*, December 2007.

[163] T. Nolte, M. Nolin, and H. A. Hansson. Real-time server-based communication with can. *IEEE Transactions on Industrial Informatics*, August 2005.

[164] S. Chatterjee and J. Strosnider. Distributed pipeline scheduling: end-to-end analysis of heterogeneous, multi-resource real-time systems. In *15th Int. Conf. on Distributed Computing Systems*, May 1995.

[165] S. Mubeen, M. Ashjaei, T. Nolte, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck. Modeling of end-to-end resource reservations in component-based vehicular embedded systems. In *42nd Euromicro Conference series on Software Engineering and Advanced Applications*, September 2016.

[166] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K. L. Lundback. The rubus component model for resource constrained real-time systems. In *International Symposium on Industrial Embedded Systems*, June 2008.

Appendices

Appendix A

Abbreviations

QoS	Quality of Service
AFDX	Avionics Full DupleX Switched Ethernet
AVB	Audio and Video Bridging
TSN	Time Sensitive Networking
FTT-SE	Flexible Time-Triggered Switched Ethernet
HaRTES	Hard Real-Time Ethernet Switching
COTS	Commercial Off-The-Shelf
FTT	Flexible Time-Triggered
DGS	Distributed Global Scheduling
RBS	Reduced Buffering Scheme
RM	Rate Monotonic
EDF	Earliest Deadline First
CAN	Controller Area Network
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
RTA	Response Time Analysis
RTCD	Real-Time Communication Daemon
EC	Elementary Cycle
TM	Trigger Message
MTU	Minimum Transmission Unit
WSN	Wireless Sensor Network
WCET	Worst-Case Execution Time
TRD	Turn Around Time
GTM	Global Trigger Message
SRDB	System Requirements Data Base

NRDB	Node Requirements Data Base
ECU	Electronic Control Unit

Appendix B

Notations

m_i	Message index i
τ_i	Task index i
SWD_i	Store-and-forward delay that is imposed by m_i
ϵ	The switch fabric latency due to hardware configuration
Λ	A set of messages
C_i	Total transmission time of m_i or execution time of τ_i
PK_i	Maximum packet size among the packets that compose m_i
D_i	Relative deadline of message m_i or task τ_i
T_i	Period or minimum inter-arrival time of message m_i or task τ_i
P_i	Priority of message m_i or task τ_i
J_i	Release jitter of task τ_i
$\mathcal{L}_i$	Set of links that m_i crosses through
$\mathcal{L}_{i,a,b}$	Set of links that m_i crosses from link l_a till link l_b
N	Number of messages in the set
l_x	link x in the network
n_i	number of links that m_i crosses in the network
$hp(m_i)$	The set of higher priority messages than m_i
$hep(m_i)$	The set of higher and equal priority messages than m_i
$lp(m_i)$	The set of lower priority messages than m_i
RT_i	Response time of m_i
$RT_{i,a,b}$	Response time of m_i from link l_a till link l_b
Id_i	Idle time calculated for m_i
$rbf_i(t)$	Request bound function of m_i at time t
$sbf_i(t)$	Supply bound function provided for m_i at time t

nP_i	Number of packets that compose m_i
O	Overhead of the Ethernet frame
MTU_i	Maximum transmission unit of m_i
$\mathcal{S}$	System notation in distributed embedded systems
L	Number of applications in a distributed embedded system
$\mathcal{A}^{(N)}$	Application number N in a distributed embedded system
$\Gamma_M^{(h)}$	Transaction number M in application number h
$\tau_{j,i,k}$	Task number k in node j belonging to transaction number i in a distributed embedded system
$m_{i,k}$	Message number k belonging to transaction number i in a distributed embedded system
$J_{j,i,k}$	Release jitter of a task in a distributed embedded system
$V_{j,i,k}$	Task predecessor information
$J_{i,k}$	Release jitter of a message in a distributed embedded system
$V_{i,k}$	Message predecessor information
$\Pi_{rq}^{(h)}$	Period of reservation on resource rq for application number h
$\Theta_{rq}^{(h)}$	Budget of reservation on resource rq for application number h
$\Psi_{rq}^{(h)}$	Priority of reservation on resource rq for application number h
$Cr_i^{(h)}$	Timing constraints for a transaction in a distributed embedded system
$D_i^{(h)}$	Deadline for a transaction in a distributed embedded system
$Age_i^{(h)}$	Data age deadline for a transaction in a distributed embedded system
$Reac_i^{(h)}$	Data reaction deadline for a transaction in a distributed embedded system