Mixed criticality management into real-time and embedded network architectures : application to switched ethernet networks

Olivier Cros

UNIVERSITÉ PARIS-EST

THESE

en vue de l'obtention du titre de

DOCTEUR

de l'université Paris-Est Marne-La-Vallée

Spécialité: Informatique

Olivier Cros

Mixed criticality management into real-time and embedded network architectures: application to switched Ethernet networks

---

Gestion de la criticité mixte dans les réseaux temps-réel embarqués et application à Ethernet commuté

---

soutenue le 8 décembre 2016

Directeur	Laurent George	ESIEE, Paris
Rapporteurs	Claire Pagetti	ONERA, Toulouse
	Christian Fraboul	IRIT, Toulouse
Examinateurs	Cherif Larouci	ESTACA, Saclay
	Rami Langar	UPEM, Marne-La-Vallée
	Xiaoting Li	ECE, Paris
Invité	Matthieu Jan	CEA, Saclay

Mixed criticality management into real-time and embedded network architectures: application to switched Ethernet networks

---

Gestion de la criticité mixte dans les réseaux temps-réel embarqués et application à Ethernet commuté

Olivier Cros

PhD prepared in ECE Paris - LACSC
Laboratoire d'Analyse et de Contrôle des Systèmes Complexes
Ecole Centrale D'Electronique
37 quai de Grenelle, CS 71520
75725 Paris CEDEX 15

PhD in collaboration with UPEMLV
Université Paris-Est Marne-La-Vallée
5, boulevard Descartes
77454, Champs Sur Marne
Marne-la-Vallée Cedex 2

PhD in collaboration with LIGM
Laboratoire de l'Institut Gaspard Monge (UMR 8049 CNRS)
5, boulevard Descartes
77454, Champs Sur Marne
Marne-la-Vallée Cedex 2

PhD in collaboration with ESIEE
Ecole Supérieure d'Ingénieurs en Electrotechnique et Electronique
2 boulevard Blaise Pascal
93162, Noisy-le-Grand Cedex

All we need is somebody to lean on

Remerciements

"C'est curieux, chez les marins, ce besoin de faire des phrases." [1]

Et pourtant, si l'on veut n'oublier personne, force est de constater que l'on ne peut pas tout boucler en une seule ligne. Parce que chacun a compté, à sa manière. Tout au long de cette thèse, chaque rencontre a été un moyen unique de s'émerveiller et de s'épanouir à nouveau. Par l'échange, par le partage, connaissances lointaines ou amis de longue date, par leur suivi quotidien ou par leur intervention ponctuelle, toutes les personnes citées ici ont été pour moi d'une importance vitale au cours de cette thèse. A défaut de tous vous remercier individuellement, ces quelques lignes vous sont dédiées.

Le jury

Il n'y a pas de thèse sans encadrement. C'est pourquoi dans ces remerciements, mes premières pensées vont à mes encadrants. A Laurent George, tout d'abord, pour avoir accepté de diriger ce doctorat. Ce fut un réel privilège de travailler sous sa direction. La rigueur et la maîtrise dont il fait preuve n'ont pas été des freins[1] mais bien des moteurs de progression, me poussant à toujours chercher d'avantage, creuser un peu plus profond. Merci d'avoir su prendre le temps d'écouter mes idées, d'avoir eu la patience proverbiale d'expliquer les mêmes choses dix fois et de m'avoir permis de mener cette thèse à son terme.

Merci à Xiaoting Li, pour ses compétences, son niveau d'exigence et son soutien. Merci de m'avoir montré qu'arriver au bout de tout cela était possible, même dans les moments les plus délicats. Merci de m'avoir fait profiter de son expertise tout au long de cette thèse. Merci enfin pour son humour[2] et sa bonne humeur permanente qui furent une source de motivation tout au long de ces trois ans.

Ensuite, je tiens à témoigner de mon plus profond respect aux membres de mon jury[3]. A Claire Pagetti tout d'abord, pour la pertinence de ses relectures. Merci d'avoir pris le temps d'identifier les aspects tout autant négatifs que positifs de ce manuscrit afin de me permettre des modifications essentielles quant à sa présentation. Mais aussi pour avoir pris le temps de faire émerger les questions laissées en suspens par mon travail en suggérant des recherches plus approfondies.

Merci à Christian Fraboul, relecteur également, pour avoir accepté de corriger le manuscrit et de m'avoir fait profiter de son expérience. Ses retours, tout aussi pertinents qu'importants, ont permis de structurer l'ensemble et de lui donner d'avantage de cohérence. Si ce manuscrit ressemble à quelque chose, c'est grâce à lui.

Merci à Mathieu Jan, Rami Langar et Chérif Larouci pour leur aide et leurs conseils dans la relecture du manuscrit et la préparation de la soutenance. Me faire l'honneur de leur présence au sein du jury représente une marque de confiance qui me touche profondément.

[1] Contrairement à ses affirmations
[2] Sur ma silhouette fine et athlétique entre autres
[3] Notamment parce que ce sont eux qui m'évaluent

A tous, merci.

Cette thèse a été une source d'épanouissement sans précédent grâce aux nombreses personnes que j'ai pu croiser au cours de ces trois ans. Merci à tous, doctorants et docteurs accomplis, enseignants motivés et professeurs motivants de m'avoir guidé par votre présence et votre implication. Vous avez tous été de grandes sources d'inspiration et d'idées. Dans le désordre [4], merci à Frédéric pour ses connaissances, sa pédagogie et pour avoir accepté d'apprendre la langue de bois avec moi, Pierre Courbin qui fut un modèle durant ces trois ans et dont les conseils m'ont permis de garder confiance en moi, Pierre Sauvage sans qui je ne connaîtrais pas les plus beaux coins de Paris, Clément à qui je souhaite toutes les chances de réussite au MIT, Thomas pour son ouverture d'esprit, son dynamisme et ses récits de voyage, Rafik pour sa finesse et sa gentillesse et enfin Tristan, sans qui je serais totalement dépourvu de culture musicale.

Pour leur soutien et leur bonne humeur, merci à Waleed et ses références cinématographiques, Houari et sa générosité légendaire, François Saïdi et son aide lorsque j'ai du commencer à enseigner, François Muller pour ses conseils sur le monde académique, et enfin Ermis et Célia dont le tour de soutenir viendra bientôt ! Merci aussi à toute l'équipe du LACSC, notamment Assia, Benaoumeur, Sebti, Jean-François, Manolo, Aakash, Jae Yun, Noura et Mustapha .

Au cours de cette thèse, être amené à enseigner fut tout autant une source d'inspiration que de rencontres. Aussi, je remercie Geoffrey, Lucie et Vladimir sans qui ARTEMIS n'aurait jamais pu voir le jour et qui m'ont beaucoup aidé à consolider les bases de ce projet.

Concevoir une thèse est difficile, mais c'est aussi la conséquence logique d'une longue phase de réflexion préalable. Durant cette phase de réflexion, de nombreuses personnes m'ont aiguillé vers la recherche et l'enseignement par leurs conseils et leur expérience. Aussi, je remercie Kévin, Jérôme, Christophe et Cécile pour leur aide à ces moments particuliers.

Faire une thèse a toujours un côté improbable, décalé, surtout aux yeux des autres. Un côté frustrant, aussi, de ne pas toujours pouvoir décrire ou expliquer les problèmes que l'on rencontre. Aussi je dois beaucoup à mes amis, dont les efforts de tolérance et de solidarité ont été bien au-delà de ce qu'un être humain normal peut supporter comme lamentations. Pour avoir enduré les crises de nerfs comme pour avoir partagé des bières et des manettes après les mauvaises journées, merci à Nicolas, Antoine-Ali, Thomas, Antoine, Camille, Myrina et Marine. Je ne m'étendrai pas sur ce que chacun d'entre vous m'a apporté car je sais que vous êtes déjà conscients individuellement d'à quel point vous comptez pour moi, en général et dans la réalisation de ce manuscrit.

Le mot de la fin

"On ne peut donner que deux choses à ses enfants : des racines et des ailes."

A Isabelle et Sébastien, et à leurs familles respectives, un grand merci. Merci d'avoir toujours été là, de m'avoir supporté. Merci de m'avoir proposé d'autres moyens de penser, d'autres moyens de percevoir. Grâce à vous, je suis un peu plus sage et un peu moins fou.

Merci à mes parents Philippe et Mireille. Merci d'avoir su me faire pousser droit, de m'avoir donné

[4] Je n'ai pas appliqué d'algorithme d'ordonnancement ici

suffisamment d'ailes pour aller loin, et suffisamment de racines pour savoir vers où revenir. Cette thèse est aussi un peu la vôtre.

Il faudrait au moins trois ans de plus pour remercier tout le monde à sa juste valeur. Pour ceux d'entre vous qui feraient partie des grands oubliés, n'en prenez pas ombrage[5], il ne s'agit que des facéties de ma piètre mémoire.

Enfin, je souhaite remercier une dernière personne. Parce que sans elle tout ce projet serait mort dans l'oeuf. Parce que je ne louerai jamais assez sa patience face à des dingues dans mon genre. Pour m'avoir dissuadé cent fois de tout laisser tomber, pour m'avoir convaincu cent autres fois de prendre du recul, de remettre à plat, de repenser les choses. Pour avoir écouté toutes mes sottises, et pour être la source d'inspiration d'une bonne majorité du contenu de ce manuscrit. Alya, merci à toi.

[5]Je peux payer une tournée pour compenser

Contents

Part I

General introduction

Chapter 1

Introduction

"Le silence éternel des espaces infinis m'effraie et la seule chose qu'on puisse lui opposer, c'est la poésie et la musique."

"The eternal silence of these infinite spaces frightens me and the only thing we can oppose to it is music and poetry."

– Alexandre Astier, quoting Pascal [2]

Contents

1.1 Résumé

La criticité mixte est une solution pour intégrer différents niveaux de criticité dans le même système au sein de mécanismes industriels intégrant des infrastructures réseau différentes. Notre objectif est de proposer des solutions pour intégrer la criticité mixte dans des domaines industriels hautement contraints afin de mélanger des flux de différents niveaux de criticités dans la même infrastructure. Cette intégration implique des contraintes d'isolation: l'impact du traffic non critique sur le traffic critique doit être borné et le plus faible possible. C'est une condition indispensable pour assurer le respect des contraintes de délai de transmission. Afin d'analyser ces délais et de centrer notre travail sur le déterminisme de ces transmissions, nous avons recours à une méthode de calcul de délai de bout en bout appelée l'approche par trajectoires. Dans ce travail, nous utilisons une version corrigée de l'approche par trajectoires, prenant en compte la sérialisation des messages.

Afin d'assurer le respect des contraintes de délai dans les réseaux à criticité mixte, nous présentons tout d'abord un modèle théorique d'intégration de la criticité mixte. Ce modèle est issu de l'ordonnancement temps réel en contexte processeur. Il présente une modélisation des flux considérant que chaque flux peut être de plusieurs niveaux de criticité.

Pour intégrer la criticité mixte dans les réseaux temps-réel, nous proposons deux protocoles différents. Le premier est le protocole centralisé. Il est organisé autour de la désignation d'un nœud central dans le réseau, responsable de la synchronisation des niveaux de criticité de chaque nœud via un mécanisme d'émission multiple fiable. Ce mécanisme est chargé de faire changer les niveaux de criticité de tous les nœuds au même instant. Le second protocole est basé sur une approche distribuée. Il propose une gestion locale à chaque nœud de la criticité. Chaque nœud gère individuellement son propre niveau de criticité interne. Ce protocole permet de préserver les transmissions d'une part du trafic non critique au sein du réseau, même en parallèle de transmissions de flux critiques.

Afin de proposer une implémentation de ces protocoles dans Ethernet, nous détaillons comment réutiliser la marque de l'en-tête de Etherne 802.1Q pour spécifier le niveau criticité d'un message directement au sein de la trame. Grâce à cette solution, chaque flux du réseau est marqué de son niveau de criticité et cette information peut être décodée par les nœuds du réseau afin d'opérer un ordonnancement en conséquence. De plus, en gestion centralisée, nous proposons une solution permettant d'intégrer les informations de gestion de la criticité directement dans les trames du protocole de synchronization d'horloge Precision Time Protocol (PTP).

Durant notre travail, nous avons conçu un outil de simulation dénommé Another Real-Time Engine for Message Integration Simulation (ARTEMIS). Cet outil est utilisé pour l'analyse de délais de transmission dans des réseaux temps-réel et pour l'analyse de scénarios d'ordonnancement à criticité mixte. Cet outil, basé sur un mode de développement ouvert et modulaire, a été utilisé tout au long de ce travail afin de valider les modèles théoriques par la simulation. Nous avons intégré à la fois les protocoles centralisé et décentralisé dans le noyau d'ARTEMIS. Les résultats de simulation obtenus nous permettent de formuler différentes hypothèses sur les garanties de qualité de service offertes par les protocoles de gestion de la criticité mixte. En termes de transmission de trafic non critique, le protocole décentralisé permet d'assurer la transmission d'une certaine quantité de messages grâce au fait que certains nœuds du réseau soient restés en mode non-critique.

Pour conclure, nous proposons des solutions d'intégration de la criticité mixte à la fois dans des contextes industriels critiques et dans des architectures Ethernet grand public. Les solutions proposées peuvent être à la fois adaptées à des réseaux synchronisés ou non synchronisés. Selon le protocole, la configuration individuelle à appliquer à chaque nœud peut être réduite afin de proposer des solutions implémentables sur du matériel moins coûteux.

1.2 Abstract

Mixed Criticality (MC) (Mixed Criticality) is an answer for industrial systems requiring different network infrastructures to manage informations of different criticality levels inside the same system. Our purpose in this work is to find solutions to integrate MC inside safety-critical industrial systems in order to mix flows of various criticality levels inside the same infrastructure. This integration induces isolation constraints: the impact of non critical traffic on critical traffic must be characterized and bounded. This is a condition to respect timing constraints. To analyze transmission delays and focus on the determinism of transmissions, we use an end-to-end delay computation method called the Trajectory Approach. In our work, we use a corrected version of the Trajectory Approach taking into account the serialization of messages.

To assure the respect of timing constraints in mixed critical networks, we first present a theoretical model of MC representation. This model is issued from MC tasks scheduling on processors. This model proposes a flow modelling which considers that each flow can be of one (low critical flows) or several criticality levels.

To integrate MC inside Real-Time (RT) networks, we propose two network protocols. The first is the centralized protocol. It is structured around the definition of a central node in the network, which is responsible for synchronizing the criticality level change of each node through a reliable multicast protocol in charge of changing the network criticality level. This centralized protocol proposes solutions to detect the needs to change the criticality levels of all nodes and to transmit this information to the central node. The second protocol is based on a distributed approach. It proposes a local MC management on each node of a network. Each node individually manages its own internal criticality level. This protocol offers solutions to preserve when possible non critical network flows even while transmitting critical flows in the network through weak isolation.

In order to propose an implementation of these protocols inside Ethernet, we describe how to use Ethernet 802.1Q header tag to specify the criticality level of a message directly inside the frame. With this solution, each flow in the network is tagged with its criticality level and this information can be analyzed by the nodes of the network to transmit the messages from the flow or not. Additionnally, for the centralized approach, we propose a solution integrating MC configuration messages into PTP clock-synchronization messages to manage criticality configuration information in a network.

In this work, we designed a simulation tool denoted as ARTEMIS (Another Real-Time Engine for Message-Issued Simulation). This tool is dedicated to RT networks analysis and MC integration scheduling scenarios. This tool, based on open and modular development guidelines, has been used all along our work to validate the theoretical models we presented through simulation. We integrated both centralized and decentralized protocols inside ARTEMIS core. The obtained simulations results allowed us to provide information about the Quality Of Service (QoS) guarantees offered by both protocols. Concerning non critical traffic: the decentralized protocol, by permitting specific nodes to stay in non critical nodes, assures a highest success ratio of non critical traffic correct transmission.

As a conclusion, we propose solutions to integrate MC inside both industrial and Commercial Off The Shelf (COTS) Ethernet architectures. The solutions can be either adapted to clock-synchronized or non clock-synchronized protocols. Depending on the protocol, the individual configuration required by

each switch can be reduced to adapt these solutions to less costly network devices.

1.3 Problematics

Safety-critical industrial domains, such as spacecraft or avionics, sometimes have to change to critical phases. These phases correspond to moments when the system increases its needs in terms of accuracy, performances and reliability. These phases correspond to delicate situations when each parameter of the system has to be very precisely managed. For example, we can cite the landing phase of a spaceship, or emergency brakes situations in public vehicles. During these situations, the system enters in specific modes, modifying its classical way of working. (MC) can be seen as a solution to represent these phases and to manage the system behavior during them.

This work proposes to integrate MC inside Real-Time (RT) networks, more particularly in RT switched Ethernet networks. In industrial domains concerned by this integration problem, the purpose is to be able to extend the MC concepts (issued from RT processor scheduling domain) to RT networks. Integrating MC answers, basically, to the following question: how to privilege and assure the transmission of specific messages in a network during critical phases? In order to provide solutions to answer to this problem, we propose to organize our work in three different levels of abstraction.

- Modelling: How can we represent and manage MC inside RT networks? Through models which have to be proven reliable, we want to extend MC concepts to networks and propose protocols and methods to manage it.
- Implementation: How can we concretely implement MC in Switched Ethernet? After designing protocols for MC management, we want to propose concrete solutions to implement these protocols inside industrial infrastructures.
- Simulation: How can we build software and hardware solutions to verify our work? We want to detail our work about RT network simulation and MC integration inside RT softwares.

In this work, we propose to detail the answers we found to these questions. All of this work is designed to answer to the following global question: What is MC applied to Switched Ethernet networks?

1.4 Outline

Part II presents the state of the art of this work. It presents all the fundamental required knowledge about RT networks we used all along our work. It is composed of three chapters. The first details the basics of RT networks and switched Ethernet. It presents the fundamental applications of these technologies inside various industrial contexts. In the second chapter, we focus on the methods to compute end-to-end transmission delays and measure timeliness and performances in these RT architectures. Eventually, in the last chapter, we present the RT simulation context we focused on. By presenting various tools and implementations, we want to propose a clear overview of RT simulation and analysis tools and the design choices they are based on.

Part III presents the different solutions for integrating MC inside RT networks. Through a first chapter, it presents what is MC and details its meaning in RT context. It presents the different solutions or concepts proposed in the litterature in processor context. Eventually, the chapter proposes solutions to transform this modelling to network context. Through the two following chapters, this part presents and details two main protocols to integrate MC inside switched Ethernet networks. These protocols, based on centralized or decentralized management of MC information, are compared through various simulation configurations in order to determine their performances and how do they satisfy end-to-end transmission delay constraints.

Part IV is about RT simulation. During our work, we designed an analysis and simulation tool called ARTEMIS. First, this part details the functional perimeter and constraints of ARTEMIS. Its design choices are detailed, and also the different architectural parameters which represent the core of the tool. Next, we present how we did adapt ARTEMIS to MC context and how we represent MC management inside the tool. The last chapter of this part presents two main modules of ARTEMIS, which can be seen as external tools not necessarily linked to ARTEMIS core itself. These parts are the topology and flowset generators of ARTEMIS. Starting from the existing algorithms proposed in processor context and detailed in the state of the art, we present here how we adapted them to network simulation context.

Part V This part concludes our work. It synthesizes the different topics we focused on and presents potential perspectives of evolution. It identifies the main point brought by this work and answers to the different problematics we had. Finally, it points out new emerging questions induced by this work.

1.5 List of publications

The Trajectory approach for AFDX FIFO networks revisited and corrected
Xiaoting Li, Olivier Cros, Laurent George
RTCSA 2014, Real-Time Computing Systems and Applications
We consider the problem of dimensioning realtime AFDX FIFO networks with a worst case end-to-end delay analysis. The state-of-the-art has considered several approaches to compute these worst case end-to-end delays. Among them, the Trajectory approach has received more attention as it has been shown to provide tight end-to-end delay upper bounds. Recently, it has been proved that current Trajectory analysis can be optimistic for some corner cases, leading in its current form, to certification issues. In this paper, we first characterize the source of optimism in the Trajectory approach on detailed examples. Then, we provide a correction to the identified problems. Two problems are solved: the first one is on the root cause of the underestimated time interval to compute delays of competing flows and a problem in the definition of the end-to-end delay computation. The second one is on the way that serialized frames are taken into account in the worst case delay analysis.

Mixed criticality over switched Ethernet networks
Olivier Cros, Frédéric Fauberteau, Xiaoting Li
WMCIS 2014, Workshop on Mixed Criticality for Industrial Systems, Ada Europe Conference
In this paper, we focus on real-time switched Ethernet networks with mixed criticality constraints. We are inter-

ested in (i) exploiting IEEE 1588 Precision Time Protocol (PTP) to implement criticality propagation techniques in such networks and (ii) analyzing delay of criticality switching. This work presents how to integrate criticality concepts for messages sent on Ethernet networks using PTP protocol. Concerning the delay of criticality switching, we consider FIFO and Fixed- Priority scheduling policies.

A protocol for mixed criticality management in switched Ethernet networks
Olivier Cros, Laurent George, Xiaoting Li
WMC-RTSS 2015, Workshop on Mixed Criticality, Real-Time Systems Symposium
In real-time industrial networks, providing timing guarantees for applications of different criticalities often results in building separate physical infrastructures for each type of network at the price of cost, weight and energy consumption. Mixed Criticality (MC) is a solution first introduced in embedded systems to execute applications of different criticality on the same platform. In order to apply MC scheduling to off-the-shelves Switched Ethernet networks, the key issue is to manage the criticality change information at the network level. The objective of this work is to propose a criticality change protocol for MC applications communicating over Switched Ethernet networks. The protocol relies on a global clock synchronization, as provided by the IEEE-1588v2 protocol, and a real-time multicast based upon it, to preserve the consistency of the criticality level information stored in all the Ethernet switches. We characterize the worst case delay of a criticality change in the network. Simulation results show how the criticality change delay evolves as a function of the network load.

Simulating real-time and embedded networks scheduling scenarios with ARTEMIS
Olivier Cros, Laurent George, Frédéric Fauberteau, Xiaoting Li
WATERS 2014, Workshop on Analysing Tools for Embedded and Real-time Systems
Real-time industrial domains are subject to strong constraints in terms of performance and reliability that directly increase the costs of their infrastructures. In order to build these infrastructures and to test them, we propose to implement ARTEMIS: Another Real-Time Engine for Message-Issued Simulation. Its aim is to manage all real-time networks like CAN or AFDX and to simulate their behaviors in terms of scheduling and performance delay. To model this tool and to make it usable, we use a modular way of development, building modules on a twoparts kernel. This architecture allows our software to be generic. Moreover, many interfaces can be easily integrated for several network implementations.

Mixed criticality management of Networked Real-Time Systems with ARTEMIS Simulator
Olivier Cros, Laurent George
WATERS 2015, Workshop on Analysing Tools for Embedded and Real-time Systems
Nowadays, providing guarantees of performances and reliability in real-time systems implies having simulation tools in order to test and emulate the systems. The real-time network infrastructures are not an exception to this rule, and needs their own simulators too. Our goal here is to present a new network simulator, ARTEMIS, which is designed to integrate mixed criticality management and real-time networked systems. Our point here is to show simulation results of ARTEMIS, especially in mixed criticality context, and to present the different main modules of this software.

Dynamic criticality management with ARTEMIS

Olivier Cros, Laurent George, Geoffrey Ehrmann
WATERS 2016, Workshop on Analysing Tools for Embedded and Real-time Systems
In this work, we propose to detail the mixed criticality integration inside our network simulator ARTEMIS. The objective here is to propose a solution to manage and simulate concrete criticality level changes inside network infrastructures, in order to focus on a network topology reconfiguration w.r.t to critical and non critical messages evolutions. Through a transmission time computation model based on a probabilistic approach, we propose a solution to generate flowsets integrating mixed criticality, in order to simulate the scheduling of these flowsets through different topologies.

FIFO* scheduling in clock-synchronized switched Ethernet networks
Olivier Cros, Xiaoting Li, Laurent George
WIP-RTSS 2014, Work In Progress Session, Real-Time Systems Symposium
For real-time applications, reliable and efficient communication on safety-critical networks has to be guaranteed for certification reasons. In order to ensure worst case transmission delay, we propose to focus on switched Ethernet networks with a particular FIFO scheduling policy denoted FIFO that requires clock synchronization. Clock synchronization is now available in off-the-shelves switched Ethernet (e.g. AVB switches). FIFO* is the scheduling based on a tag in the frame containing their release time at their source node. We want to explore the benefits of FIFO* compared to classical FIFO scheduling in term of end-to-end response time and for providing reliable multicast services.*

Chapter 2

Concepts and notations

"Quand tout semble aller contre vous, souvenez vous que les avions décollent toujours face au vent."

"When everything seems to be going against you, remember that the airplane takes off against the wind."

– Henry Ford

Contents

2.1 About Real-Time

2.1.1 What is Real-Time?

"Real-Time (not comparable): computing of a system that responds to events or signals within a predictable time after their occurrence; specifically the response time must be within the maximum allowed, but is typically synchronous." [3]

In the common sense, the expression of RT is often understood as a synonym of "dynamic" or either "simultaneous": the different actors of a real-time event are synchronized. In daily life, when things are done 'in real-time', it usually implies the concept of not being interrupted, and being checkable at any time. It also implies for things, tasks or jobs, to be finished in a finite and observable time. When the television says us 'This match will be streamed in real time', we understand the notion of 'we will see at home what happens on the playground at exactly the same time'. In fact, 'in real-time' implies things to be simultaneous. Real-time, in that case, can be understood as 'on-live'.

Of course, as we (computer scientists and scientific researchers) like to complexify all the simple things in life, we propose more detailed and complex definition to it. Executing a job in real-time means that we are able to specify the date when it starts, the delay it will long and to observe if it will be finished before a precise time (that is what we call a 'deadline'). We do not just want to execute things, but to be able to compute the time it takes, and the dates of start and stop.

Doing things 'in real-time' does not necessarily mean doing things as quickly as possible. It is a common error to mix "performances" and RT. If we want to employ a common metaphor, RT does not mean to answer to the question of 'how fast can you run?' but more of 'How can you guarantee me that, even if you are sick, injured or upset, whether it is rainy or if the ground is wet, you are sure you will be able to arrive before this precise instant? How can you assure me that there won't be a day, at any time or specific circumstances, where you won't be able to run correctly and not assure your guarantees?". My job, as a young pretending PhD in real-time, is (first of all) to modestly try to answer to this question. That is why I need to start by the beginning with this question: what is real-time?

According to [4], a real-time system is the combination of 'real' and 'time':

- 'real' means that the reaction of the system to external events must happen at the time when events happen. Even if there is a delay or an additional waiting between an event and its consequence, this delay is quantifiable and can be evaluated and bounded.
- 'time' means that the correctness and precision of the system does not only depend on the system state, but also of the system state at a given date.

We extracted the following example from the litterature [5]: We suppose a car, waiting for a traffic light at a crossroad. The traffic light can be either green, or red. Observing the traffic light system from the point of view of real-time will consist in adding the time as valuable information. The question to ask will no longer be "Is the traffic light red or green?" but "how long will the traffic light stay green?". At a given instant, the light can be green, it does not present a sufficient solution to consider the system in its globality.

In order to assure the safety of the car's driver and people passing by the crossroad, all the timing evolution of the traffic light must be analyzed, and all red and green-light periods have to be time-bounded. The traffic light can be considered here as a real-time system.

2.1.2 Real-Time Systems

A RT system is a system (vehicle, computer, aircraft, ...) or an infrastructure (embedded network, electronical board, microcontroller chipset, ...) having to respect timing constraints. Assuring the integration of these constraints inside the design and design of a system is the purpose of the RT analysis. As a deduction, RT is basically the domain of constrained computing. The RT domain consists in analyzing, testing, verifying and certifying systems at different levels in order to integrate and validate the respect of the defined constraints. Its design is to test, assure and analyze performances of hardware and software systems for the purpose of reliability, safety, security and performance of each subsystem inside a global architecture (for example, the reliability of mechanical functions inside a personal car). Each RT system can (and has to) be certified at different levels of certification depending on its needs and purposes. The higher importance we attach to a subsystem, the higher certification level it needs. For example, the safety constraints in a car will require high certification levels on brake design and control while our expectations on less important functions like the air conditioner will be lower.

Defining a RT system corresponds to integrate the pre-defined constraints into the design and design steps during its building process. When it comes to define these constraints and their sources, it can be of different nature: costs (managing the best performances with the cheapest materials), space and weight (reducing the size needed by the system, for example for avionics purposes), energy consumption (optimizing and reducing the energy cost of the system), etc... The point of RT design is to be able to make systems respect their constraints while being manageable and affordable.

2.1.3 Domains: What is it good for?

In most of safety-critical domains, particularly those with high needs in terms of reliability and safety, the costs introduced by errors can be dramatic. There is no need for anyone to detail the problem represented by the human and financial cost when an error occurs in the mechanical controls of an aircraft. As a result, in order to satisfy the constraints and assure reliability and safety inside their systems, safety-critical domains imply defining systems which are offering levels of guarantees and certification in terms of error management, detection and in assuring the respect of constraints inside these systems. That is the purpose of defining RT systems: building systems and concepts to offer guarantees in terms of timing constraints respect and integration.

Integrating RT models and processes inside a system increases its cost and complexity of implementation. It implies defining and using specific tool licences, to increase the test potentials, to find experts dedicated to substystems analyses, to certify the techniques, to assure the models, etc... That is why the use of RT systems has to be justified by the necessity of offering guarantees and precise evaluations to prevent errors, failures, and to be able to estimate the value of these errors. The higher the cost of an error, the more a system is constrained in its size and infrastructural cost, the more it is going to need

RT implementations and processes.

The main domains where RT is basically the mostly used are safety-critical ones, like defense (transport, trajectory anticipation, automatic piloting, ...), public transports, smart cars, Internet of Things (IoT), but we can also mention avionics or even aerospace.

Especially with safety and reliability constraints, RT has been proved to be proved correct by construction. In order to make the systems able to respect the needed constraints, we need to define infrastructures and models satisfying the constraints. But RT is not just a matter of safety and reliability. In order to satisfy objectives of performance in terms of space, weight, costs or energy consumption, we need to implement RT models inside systems.

The goal of RT systems is to be able to manage wide quantities of data in short amount of time and to guarantee that each task will be executed on time, conformly to specifications and constraints. In infrastructures constrained in terms of weight, spaces and energy, integrating these constraints often implies defining two different problematics: First, we need to compute the size of the needed infrastructure we have to use, and then we have to set parameters for the behavior of the different elements in the system in order to respect the constraints. RT constraints appliance does not just imply to drastically increase the size of an infrastructure, but more to learn how to adapt the correct infrastructure and protocols to the needs of the system.

RT can be applied to various industrial domains such as defense (Thales), avionics (Airbus), aerospace (Nasa, TTTech), robotics, etc...

2.2 Real-Time scheduling

The purpose of RT scheduling is to focus on timing constraints management into systems. RT scheduling regroups the list of methods, solutions and models to manage task executions in a system to be done on time, w.r.t. to various constraints [6]. Usually, constraints inside a system are defined to respect guarantees and performances purposes. This allows the system to be conform to different standards and specifications. Depending on the context of use, these standards and external constraints could be defined by different sources. We can mention ARINC for aeronautics, for example, which is defined and maintained by various actors of the aircraft domain (Boeing, Goodyear, General Motors, etc...).

2.2.1 Network and processor context

In this work, we operate a split between two different subdomains of RT: processor context and network context. The major part of our work is focused around network context, but some basic definitions we need come from the RT processor context. We introduce here these two contexts.

Processor context

A RT system is composed of a physical device made of one or several cores and clusters [7], each one composed of one or several processors responsible for executing all the tasks of a system. For example,

a car's dashboard has to display regularly the vehicle speed, while computing the current fuel consumption and displaying all the needed alerts to inform the driver. The dashboard is composed of different elements which send orders and tasks to do to a central unit. This central unit is a set of several processors (organized in cores) and is responsible for executing all the ordered tasks.

Focusing on the scheduling of all the tasks inside a set of cores and processors is the purpose of processor-oriented RT scheduling. This domain focuses on the execution of tasks during a given time interval with different timing constraints.

In terms of modelling, constraints of performances and guarantees can be represented differently in processor-context: either it is constraints about tasks (preemptivity, deadlines, activation instants, ...) or constraints imposed to the platform (scheduling policies, architectures, materials, memory management, ...). The purpose is to focus on the timing analysis for a given taskset on a given architecture and to analyze the timing results provided by such an execution.

RT scheduling has been studied first in a uniprocessor and multiprocessor contexts. These contexts are the contexts of performance and relability managing inside systems integrating embedded processors and chipsets with a central memory. On the opposite, RT networks is a domain based on communicating systems with a distributed memory. In terms of RT basic definitions, both processor and network contexts are submitted to the same models.

In processor context, we represent a system as a set of tasks to execute on a given pre-defined platform (a device composed of cores and processors, all submitted to various implementation constraints). For example, a personal computer can be represented as a set of 50 tasks to execute, each task linked to one or several softwares. These tasks can be: referesh the screen, launch the web browser, display a message, etc...

Network context

A RT global system like a car or an aircraft is composed of different elements communicating with each other. For example, a car is composed of sensors (wheel position, speed, direction, temperature, ...) which directly transmits informations to computation units through network intermediate commuters (called switches). Each sensor, each command can be seen as input or output for the network and we can define one or several information flows between the same sensors.

Definition: Node

A node represents a data transmission device in a network. In can be an end-system (sensor) or a switch.

When it comes to integrate and model constraints in these systems, we focus on message transmission. The purpose is to understand how to transmit information from an end-system to another, and to compute the end-to-end delay needed for a message to transit through the network, defined as a set of interconnected nodes. This part of RT scheduling is the network-oriented context.

There is many distinctions between these two branches of RT scheduling which we specify all along this work. But we can basically synthesize that the purpose of RT scheduling in processor context is to

execute a set of tasks in a static computation unit, whereas RT network context consists in transmitting information through different computation units linked by physical links.

In network context, task execution is no more relevant: we do not consider anymore a machine as a processor (or a set of processors). Our work is focused on integrated network topologies composed of a set of end-systems communicating and transmitting messages with each other. These communicating machines can be sensors, displaying screens, user-oriented commands... They are all parts of the network, defined as its input our output points, called end-systems. All the end-systems of a network are linked with physical wires to commuters (network switches) whose role is to route information from their source end-system to their destination end-system. Each of the nodes (end-systems and switches) can also be called nodes when we do not need to differentiate them.

Each electronical command (brakes, paddle, headlights command) and each sensor (speed, pression, temperature) is represented as a node. If it measures and sends informations (sensor), it is considered as an input node. On the contrary, if it is configured to receive information (screen, Light-Emitting Diode (LED)), it is called an output node. In our network modelling, we consider these input and outputs nodes as end-systems. The path of a message should start and end in an end-system. Basically, we consider that an end-system can both transmit and receive network flows.

2.2.2 Timing analysis

For both network and processor contexts, RT scheduling focuses on how to schedule events corresponding to task releases or message transmission requests on time. In order to analyse these scheduling problems, we introduce different fundamental concepts we will use all along this work. In processor context, each task is defined with different parameters. The basic parameters defining a RT task τ_i are:

- The worst case execution time (WCET) C_i: each task needs a specific time to be done. The execution time of a task is precisely the time needed to run it.

- The period T_i: A task is not necessarily unique and can be released several times over time. Each release of a task is called a job. Sometimes, the release time of a job can be randomly defined, sometimes it is constrained by a minimum inter-arrival time.

- The deadline D_i: We distinguish relative and aboslute deadlines of a task. Deadlines are the most pertinent representant of timing constraints applied to a RT system. We have the relative deadlines, first, which represent the maximum delay between the release date of a job and its execution end. A relative deadline can be defined according to different models. We have deadline on requests ($D_i = T_i$) when job execution should be finished before the release time of the next job, constrained ($D_i \leq T_i$) or abitrary when there is no constraint between T_i and D_i. Secondly, we note the absolute deadline, which is the latest date at which a job has to be finished.

These concepts are illustrated by the figure 2.1.

In order to manage and prioritize the different jobs of a task, we can apply one or several task management policies called scheduling policies. There exist many different scheduling policies defined in RT

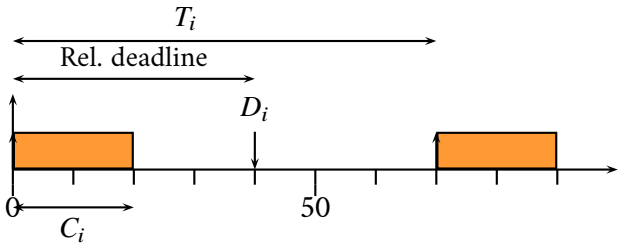


Figure 2.1: Example of a periodic task

The purpose of RT scheduling is to provide models and scheduling policies to test and assure the schedulability of a system. A system is said schedulable if each task is able to respect its deadlines given the different constraints applied to the system.

scheduling. We can mention famous ones like First In First Out (FIFO), Fixed Priority (FP) (arbitrary-defined priorities), Rate-Monotonic (RM) [8] (priorities computed based on period), Earliest Deadline First (EDF) [9] (priority based on absolute deadlines), etc...

When we focus on deadline definition, there are two different approaches in RT scheduling. Either we consider that there is no possible deadline miss in the system (Hard RT) or that deadline miss is possible and has to be characterized in terms of impact and costs (Soft RT). We can also consider a potential mix of these two hypotheses, depending on the task. For example, in an airplane, executing braking on time is a hard-defined constraint, whereas periodically informing the passengers can suffer from some slight delays or deadline misses. Introducing soft RT does not necessarily mean that all deadlines can be exceeded.

In this work, we consider that we are working in hard RT context: we do not consider that a deadline miss is acceptable.

2.2.3 Transmission time

Worst case analysis

Each job from the same task may have a different execution time, varying according to different parameters. For example, depending on the day, doing the dishes will take me 20, 25, 40 or 10 minutes. These are the different execution times provided by myself for the task "do the dishes".

In RT scheduling analysis, providing guarantees and reliability inside a system implies defining systems which are proved to be reliable and stable in any execution case. It means that RT scheduling analysis has to be based on a model which takes into account all potential execution times. We base our work on a worst case approach of each task. It means that we do not consider the real execution time of each job, but the worst case one each time (in that case, 40 minutes is the real worst case execution time of my task).

We have to provide guarantees of reliability on task scheduling. It means that we consider that, at each execution, a task took the maximum possible time, called the Worst Case Execution Time (WCET). Modelling and computation of the WCET have been presented in works like [10], [11] and are more detailed below in 2.2.3.

According to the circumstances, a task execution time can vary. It can be slowed by electrical and

electronical performances of the machine, submitted to several external parameters. It means that, during repetitive executions of the same task, the obtained execution time cannot be the same each time. In order to obtain WCET computations based on experimental results, we use specific algorithms [12] to extrapolate the obtained results and establish reliable WCET values (see figure 2.5).

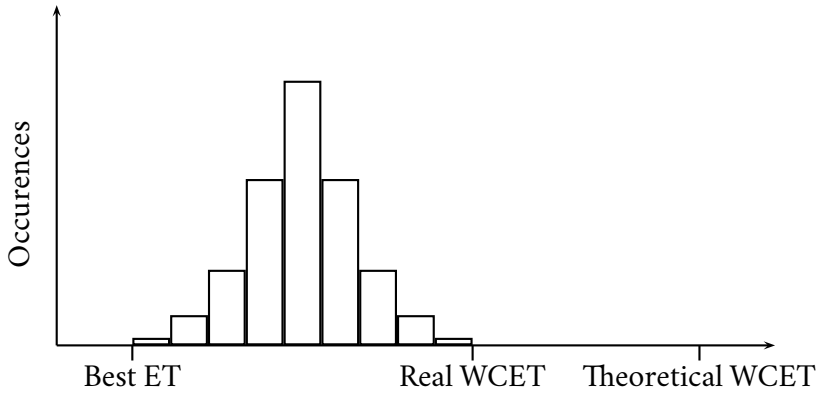


Figure 2.2: WCET computation model of a task

In order to model the worst-case execution time of a task, we introduce the concept of WCET in processor context. The WCET of a task is computed according to its execution profile. It corresponds to an execution time that will never be exceeded by the task execution at run time. This worst case approach introduces pessimism in delay and performances computation, but it allows us to affirm that, if a system is schedulable in the worst case situation, it will be schedulable in all execution cases if it is sustainable w.r.t. WCETs [13].

Given the execution profile of a task, we can deduce its real-WCET, based on an experimental observation of its execution. But, in order to be reliable, the expression of a WCET has to be computed and guaranteed and not just based on empiric experimentation observations. Works like [11] proposed mathematical models to compute task WCETs.

But this theoretical computation of WCETs introduces a pessimism between the theoretical WCET and the real WCET of a task. This gap has been treated in different works like [14]. This gap introduces pessimism in the timing analysis of a system [10], [15].

This pessimism is illustrated in figure 2.5. We observe in the figure a clear difference between the real WCET (computed based on a probabilistic cumulative expression of a set of executions of the same task) and the theoretical WCET, computed through mathematical models.

2.2.4 Flows activation model

In this work, we introduced the concept of period, corresponding to the delay between two activations of the same message. This period model is only viable when messages activations of a flow are periodic. In RT scheduling, the task activation model is a concept that needs to be properly defined. We present here major activation models: periodic, sporadic and aperiodic. This presentation is based on previous works done on activation models [4].

Network modelling consists in a set of flows sent through static paths inside a network. Each flow produces messages, consisting in a set of bytes, following different Media Access Control (MAC) network protocols (Ethernet, for example). This is the basics of message emission in the RT network modelling.

But emitting messages from a source node can be done according to different models. Each flow can produce several messages, and the activation model between messages has to be precisely defined. In the RT paradigm, there exist plenty of possible activation models: periodic, aperiodic, sporadic, strictly periodic, according to a scheduling table, etc...

In our work, we consider different possible activation models of messages. We integrate specifically these because they correspond to the implementations we can find in the network architectures we focused our work on. We consider different potential activation models, detailed as follows.

Periodic

In periodic model, the time interval between two successive messages emission requests (from the same flow) is constant. This time interval is called a period. The period is defined by the system designer during network implementation. See [16], [17] for details about RT scheduling of period tasks. According to the period, we can determine the flow realease times among time (see figure 2.3).

Sporadic

The sporadic flow activation model means that there is no identifiable periodic behavior in messages emission requests from a flow. But we can guarantee a minimum delay (starting from a message sending instant) during which there is no additional message emission requests. This delay is called the minimum inter-arrival time of the flow (see figure 2.3).

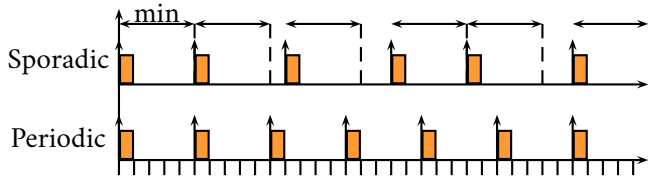


Figure 2.3: Sporadic and periodic flows activation models

We consider that each flow produces messages as often as possible, generating the highest quantity of traffic it can. As a result, the minimum inter-arrival time of a flow can be assimilated as the period of a sporadic flow. Sporadic flows are often considered as periodic flows in the worst case analysis. For further details, the sporadic model was detailed in [18].

There exists different other activation models, such as aperiodic [19]. They will not be considered in this work.

2.2.5 Preemption

The hypothesis of the preemption or non-preemption of the transmission of a message consists in answering the following question: is it possible, during a message transmission inside a node, to stop its transmission, start transmitting another message, then resume the transmission?

The first hypothesis in RT network context is about the non-preemptivity of a message transmission. It means that once the transmission of a message has been started in a node, it cannot be interrupted in that node.

Imposing the non-preemptivity of a message transmission induces that there is no partial emission of a message. There is no risk of potential reception of a incomplete frame, and we do not have to consider the potential transmission of a malformed frame. Each frame is necessarily compliant to the different tags induced by the network standards we defined. We consider non-preeemptivity as a fundamental hypothesis in all our work.

Figure 2.4 shows a simple example of preemptivity in processor context. If we suppose two tasks τ_1 and τ_2 defined by a respective WCET of $C_1 = 15$ and $C_2 = 20$ μs and a respective priority of 0 and 1 (τ_2 has a higher priority).

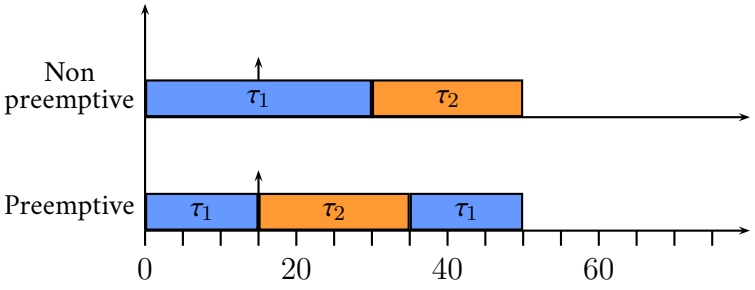


Figure 2.4: Non-reemptive and preemptive tasks

We consider that τ_1 activates at $t = 0\mu s$ while τ_2 activates at $t = 15\mu s$. The system is composed of a unique processor scheduled with FP policy. We have an illustration of preemptive and non-preemptive scheduling in the results shown in figure 2.4.

2.3 Modelling

RT scheduling applied to networks consists in characterizing the worst case end-to-end transmission delay of a message in a network topology, starting from its source node to its final destination node. To compute this delay and being able to propose reliable methods to guarantee it, we first need to define the network model we use all along this work.

We consider a network $\mathcal{N}$ defined by a topology and a set of flows. The topology is represented by a set of end-systems $ES_1, ..., ES_{n-1}, ES_n$ and a set of switches $S_1, ..., S_{m-1}, S_m$. In order to assure the continuity of the network, each switch and each end-system in the topology is connected to at least one another node.

In our modelling, we also have to introduce the modelling of flows. A flow v_i is considered as a producer of messages with a periodic or sporadic model. We note a flow period T_i. Periodically or sporadically, a flow v_i produces a message m_i of a specific size in bytes. Each produced message is supposed to have a specific size but, as we are working on worst case analysis, we consider that each produced message has the longest possible size, defined by the properties of the flow.

Each message m_i from a flow v_i follows the same statically defined path. The nodes path of flow v_i is denoted as $\vec{\mathcal{P}}_i$. This path is acyclic (not the same node twice). It is composed of an end-system (as source), as set of switches and, eventually, another end-system.

Definition: Worst Case Analyzing Time

The WCAT of a message is the ratio between its size and the network bandwidth.

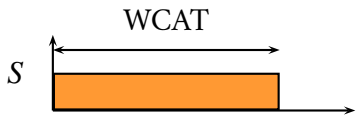


Figure 2.5: WCAT of a flow

In the following work, we make the assumption that the WCAT of a message is the same in all nodes of the network. This is based on the hypothesis that a network has a global bandwidth.

Definition: Flow

A flow is a set of messages, periodically or sporadically produced. All these messages are dedicated to the same functional purpose. A flow is characterized by the 3-tuple : $\{\vec{\mathcal{P}}, C_i, T_i\}$ *(path, WCAT vector, period).*

Each flow is characterized by a path of nodes, going from an end-system to another one through a set of switches. In order to guarantee the determinism of transmissions, we do not rely on an automatic calculation of a flow path. It means that we suppose, all along our work, that each flow path is statically defined by the network designer.

Each flow produces messages of a specific size. The WCAT of a message is computed on ratio between the message size and the network bandwidth. Each message is conformed to specific network standards (in our case, Ethernet) which means that each message is bounded in terms of minimum and maximum size (and, as a result, in terms of minimum and maximum WCAT).

2.4 Conclusion

Considering the different definitions we made, we assume the following hypotheses in our work :

- Each message has a maximum size
- A flow can be either sporadic or periodic
- The path of flow through the network is statically defined by the designer
- There is no potential transmission error in the network, neither in the links or in switches
- All network links are considered as full-duplex

Part II

State of the art

Chapter 3

Real-time Networks

"Je sais que je ne sais rien"

"As for me, all I know is that I know nothing"

– Socrates [20]

Contents

3.1 General considerations about real-time networks

One main characteristic of a RT network is its determinism [21]. A RT network is called deterministic if each transmission time of each message in the network can be bounded. In our work, concerning network modelling, we consider the following hypotheses :

About the network structure, we consider :

- The network architectures we consider are offering full-duplex connections. It means that we do not consider collisions nor collision management protocol integration inside the network architectures we focus on. Collision management protocols such as such as CSMA/CA [22], [23] for Wi-Fi, CSMA/CR [24] and CSMA/CD [25] will not be detailed in this work.
- There is no redundancy in messages transmissions. If a message transmission has failed or if the message is dropped out of the network, there is no potential retransmission of this message in the network.
- The networks we consider do not integrate fault-tolerant mechanism.

3.2 Ethernet-based solutions

3.2.1 Introduction

The purpose of our work is to integrate MC inside RT network architectures. In order to have an exhaustive point of view of what had been done in terms of RT integration, we first need to present the different network architectures we based our work on. All the network architectures presented below are based on an Ethernet infrastructure.

Our work is based on a the analysis of two different contexts of RT Ethernet implementations. First, we focus on industrial-oriented technologies (Avionics Full DupleX switched Ethernet (AFDX), Time-Triggered Ethernet (TTEthernet)). These technologies are costly and introduce a wide set of implementation constraints (availability of the materials, requiring a high degree of expertise, ...) but they offer a very high level of potential configuration of the different devices. Second, we focus on open COTS-oriented architectures, which privilege the costs but offer a relatively poor potential of configuration. Our purpose is to present both these approaches through different Ethernet implementations, in order to provide a general description of RT networks potentials.

3.2.2 Commercial Off The Shelf Switched Ethernet

What is COTS switched Ethernet?

In the industrial domains concerned by RT networks and subjected to strong implementation constraints, network protocols have to be proved reliable. In order to satisfy their functional needs and offer their

safety and reliability guarantees, each protocol has to be certified and proved adequate to various standards. That is this context which led to build dedicated network infrastructures such as TTEthernet, Audio-Video Bridging (AVB), Controller Area Network (CAN), etc...

Implementing these industrial network infrastructures requires specific physical devices. This implies a strong problematic of costs and availability. That leads to the following question: What about the implementation of RT network protocols in cheapest network architectures?.

The first question that comes in mind is "Why should we?". Even if ressources costs are a burning issue in every system design, we can make the hypothesis that defense or spacecraft domains can afford dedicated budgets even for specific network infrastructures. But when it comes to domains such as home automation, personal vehicles or public transports, increasing the costs of network infrastructures in such domains can have a direct financial impact on public applications. That is the purpose of COTS infrastructures: proposing reliable and efficient network protocols inside simple physical devices which can be massively produced at a reduced cost.

In our work, we consider COTS Ethernet as a very important architecture to focus on. Beyond the problematic of costs, designing a protocol compliant with COTS Ethernet induces more genericity: if a protocol is compliant to COTS Ethernet, it will be compliant to all the more specific protocols based on it. COTS switched Ethernet devices provides several advantages compared to specific network implementations:

- The devices are spread among all application domains, the commercial availability of the devices is not a problem and the delays to obtain it are short.
- It is a generic solution, implemented for a long time. As a conclusion, is has been proved reliable through the time by different kind of services.
- Commercially speaking, costs are reduced also by the potential concurrence between the different manufacturors. On the contrary, specific solutions belonging to a dedicated industry allows them to keep a commercial monopole.

Standard details

The cheapest network Ethernet standard is the public commercial version of Ethernet: IEEE 802.3 [26] standard. It is the common network implementation of the ISO model we can find in many public and private implementations (personal houses, companies, administrations, etc...). It integrates the 802.1 standard, and especially the Local Area Network (LAN) specifications and MAC addresses management. Each device is attached to a dedicated MAC address, used for forwarding the messages inside the network.

IEEE 802.3 specifies the standard for Ethernet networking. It concerns the physical and link layers of the OSI model. Concerning the physical layer, a COTS Ethernet network is represented as a set of interconnected devices. In this work, we make a difference between end-systems (computer, sensor, ...) and network-management dedicated devices (switches). Each device is connected to others through copper or fiber wires.

Concerning the link layer, IEEE 802.3 specifies different needed elements to establish a network connection through an Ethernet network. This layer is responsible for message transmission. Particularly in RT networks, proving the timeliness of the transmission is a strong constraint. It means that the end-to-end transmission time of each message in the network has to be upper-bounded.

There is a wide subset of standards which are based on IEEE 802.3, improving it in terms of reliability, bandwidth or structure. Depending on the architecture, IEEE 802.3 and these subsets can provide a panel of different bandwidths. COTS Ethernet is based on a 100 Mb/s bandwidth, but this bandwidth can be increased up to 10 Gb/s in such protocols like 802.3ae [27].

Commercial Off The Shelf Ethernet frame

In Ethernet frame, datas are encapsulated in a formatted frame. This frame contains all the needed informations to send and forward the message from its source node to its final destination node. We consider that a message data is a set of bytes (maximum size: 1500 bytes). The IEEE 802.3 LLC frame format is detailed in figure 3.1. We can split this frame in distinct parts:

- A preamble of 8 bytes, which is not directly part of the frame.
- A header, composed of 20 successive bytes containing: the MAC source and destination addresses of the message, and the type of the message. Several Ethernet-based protocols (like 802.1Q) will modify this header by adding bytes, but basical IEEE 802.3 header is composed of this 20 bytes. The structure is detailed in figure 3.1.
- A 4-bytes suffix: a frame-check sequence (called the CRC) and an interpacket gap of 12 bytes.
- Two successive frames are separated by a silence of emission of a 12 bytes size.

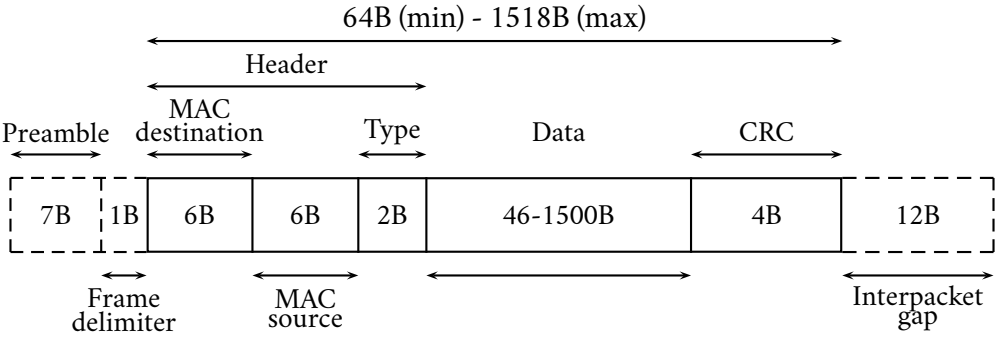


Figure 3.1: Ethernet 802.3 frame

Figure 3.1 details the structure of an IEEE 802.3 Ethernet frame. The data field size can change from one message to another, but all the other fields sizes are static, and the structure cannot be changed. The different fields composing the header and the suffix have all their specific role in Ethernet forwarding, detailed as follows:

- Preamble: This set of 7 bytes is composed of the byte pattern repeated seven times. It corresponds to an alternated succession of 0 and 1 on 56 bits. It is not strictly a part of the message, but it is needed by the switch for eventual clock-synchronization.
- Frame delimiter: This symbolizes the beginning of the frame itself. The frame delimiter is built in order to break the preamble alternated pattern. The value of the frame delimiter is always equal to $0xD5$ (10101011, or 171 in decimal).
- MAC destination: This is the MAC address of the final destination node. It can be a unique identifier of the physical network interface attached to the destination of the message (unicast context [28]). In other cases (broadcast, multicast), this can identify a whole group of devices. More details about the MAC address formatting and representation are given in [29], [30]. In IEEE 802.3 Ethernet, this address is represented with 6 bytes.
- MAC source: This is the MAC address from the node that emitted the message. Like the destination address, this is a unique identifier of the physical interface of the node, encoded on 6 bytes.
- Type: This represents (on 2 bytes), the type of Ethernet protocol used.
- CRC: This field is used for error detection and integrity control. The computation of this field is detailed in [31]. It is encoded on 32 bits (4 bytes) and its value is based on the message content.
- Interpacket gap: This is an idle period, without any transmission. This period is constrained in its minimum duration by the bandwidth of the network (0.96μs for 100 Mb/s Ethernet). This corresponds to a 12 byte size gap. This silence introduces a recovery time for each message before preparing to send the next message. This gap allows the device to recover its clock.

Ethernet frame format implies each frame size to be upper and lower bounded in terms of byte size. The minimum size of a message happens when the data field is equal to its minimum size 64 bytes. This corresponds to a minimum data payload size (46 bytes). In the case we want to send a message shorter than 46 bytes, we put padding bytes at the end of the payload, to complete it up to 46 bytes. On the contrary, the maximum size is reached when the data field is equal to 1518 bytes. These limits do not take into account preamble, delimiter and interpacket gap fields, which are not directly part of the frame or data to send, but are just dedicated for bit rate synchronization.

Given the direct relation between the size of a message and its WCAT, these limits of size means that in Ethernet networks the WCAT of each message is bounded and depends of the bandwidth. As a conclusion, in a classical IEEE 802.3 100 Mb/s bandwidth, a WCAT is bounded between $5,1\mu$s ($\frac{64*8}{100*10^6}$) and 121μs ($\frac{1518*8}{100*10^6}$) (see [32] for details). This constraint has to be taken into account, specifically for simulation purposes when the bandwidth has to be indicated.

VLAN management

One main problem in RT Ethernet infrastructures is to be able to provide high performances with strong constraints of space occupation. In fact, the size of an aircraft is constrained and so are the different

subsystems inside it. The quantity of wires and network devices to install inside the airplane has to be accurately defined and constrained.

On the reception port, an end-system associates a waiting queue to each Virtual Local Area Network (VLAN), in a statically-defined order of priority, and then applies a FIFO scheduling policy to each queue. Each time an end-system receives a message, it is in charge of controlling if it is not redundant. In order to integrate VLAN management and identification, Ethernet proposes the 802.1Q frame format. It is an improved alternative version of Ethernet 802.3 (see 3.2.2), built for VLAN support and integration. The point is to add 4 new bytes to Ethernet header, after the MAC source address. This tag identifies the VLAN a message belongs to. The structure of these bytes is detailed in figure 3.2.

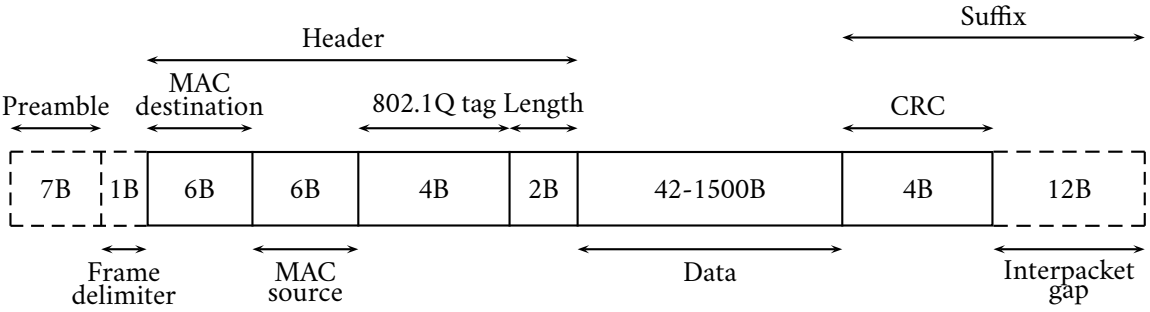


Figure 3.2: Ethernet 802.1Q frame

The 4 bytes of Ethernet 802.1Q are splitted in different parts:

- TPI: Tag Protocol Identifier. The first two bytes of the Ethernet 802.1Q header allows to specifically tag an Ethernet frame as 802.1Q compliant, adding the $0x8100$ tag to it. These two tags are used only for 802.1Q identification.
- Priority Code Point (PCP): This set of 3 bits is used to attribute a dedicated priority value to each message, from 0 to 7.
- Drop Eligible Indicator (DEI): It allows to tag a message to decide whether it can be dropped out of waiting queues or not in case of network congestion.
- Tag Control Information (TCI): The two last tags of 802.1Q header are used for priority management and VLAN identification. These bytes are splitted as described in figure 3.3.

The TCI field of Ethernet 802.1Q is composed of 16 bits, splitted in 3 different fields. The first field (3 bits) is the PCP field. It associates a dedicated priority value to each message.

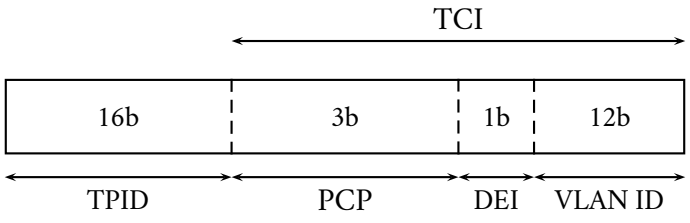


Figure 3.3: Ethernet 802.1Q tag details

Each message is associated to a priority from 0 to 7. It means that even in the same VLAN, messages can have different priorities.

The second field is the DEI (1 bit). It is a indicating if the message can be dropped out of the network in case of congestion. It is a first solution to tag a message if it can be ignored or if it has to be transmitted to its destination even in case of high network traffic.

Finally, the 12 last bits of the TCI field are used to store the VLAN identifier of the message. In AFDX and 802.1Q-based networks, each physical network can be subdivided in a maximum of $2^{12} = 4096$ different VLANs, each one identified by a unique integer. The VLAN identifier allows to specify to which VLAN the message should be sent to. In association with the PCP field, the VLAN identifier allows us to integrate priority management in the network.

3.2.3 AFDX architecture

General concepts

Avionics is a safety-critical domain requiring specific RT networks standards [33]. AFDX (Avionics Full DupleX) is a switched Ethernet network protocol first designed for avionics purposes, and especially for Airbus, embedded in airplanes and aeronefs. Its first (and currently used) version has been designed by Airbus in 1998 in compliance with the ARINC 664 chart [34] as a new RT standard for avionics communication. AFDX was the part 7 of ARINC 664 and was finally published in 1999. The job on AFDX started from the lack of performance of ARINC 429 [35]s, whose bandwidth was limited to 100Kb/s. When designing ARINC 429 and then, ARINC 664, the first main constraint was to define a new communication standard based on reliability and performance of data exchange, especially in terms of error control. Like ARINC 429, ARINC 664 was only designed for internal communication between elements which are already part of the aircraft. ARINC 664 standard does not cover communications with external which are not a direct part of the aircraft.

AFDX is designed for reliable RT communication. As a conclusion, it has to be certified at the highest level, specifically to assure deterministic delay computations and precise integration of time constraints in the system. Based of these constraints, the main purpose of AFDX was to assure reliability of the network communications.

AFDX internal network architecture is based on COTS Ethernet switches. The AFDX protocol emerged then from two different technologies combination: Ethernet 802.3 (for network forwarding) and Asynchronous Transfer Mode (ATM) (for communication). AFDX is a reliable RT network protocol designed for high-constrained industrial applications, and based on low-cost COTS infrastructres.

Application domains

AFDX standard was initially the property of Airbus who implemented it in long-courrier aircrafts (A380) which had more performance needs than the smaller A320. Nevertheless, the structure of AFDX, which is based on standardized network protocols and COTS switches, allows it to be portable and potentially implemented in a large panel of infrastructures requiring RT constraints, without needing clock-synchronization.

Topology management

AFDX is a full-duplex link (contrary to basic Ethernet 802.3 which is half-duplex). This full-duplex standard allow the netword connection to not having to face with collision management problems.

In order to avoid the problem of collisions without breaking determinism constraints, each AFDX link between two nodes is full-duplex. It is defined with a twisted pair (T_x for transmission, and R_x for reception). As a conclusion, each link (T_x or R_x) is defined for a specifical way of communication and there is no possible collision (see figure 3.4).

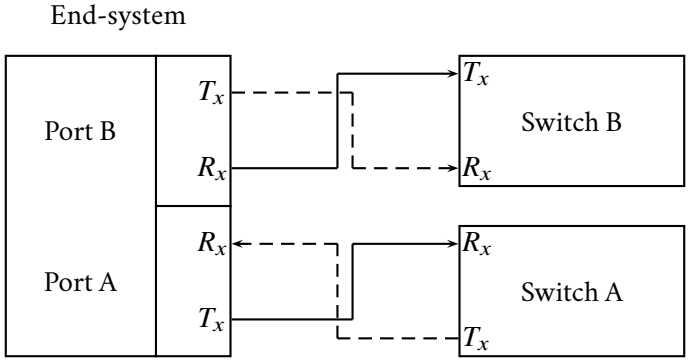


Figure 3.4: AFDX full-duplex implementation

Each defined VLANs relies on a system of specific max bandwidth allowation (based on global bandwidth). This dedicated bandwidth is computed with the end-systems by a system of contracts, applied by each emitting end-system of the VLAN. Thus, each message is limited to a maximum size, and the time interval between two messages is constrained. These different constraints allow us to maintain a maximum transmission rate and network traffic in each VLAN, to assure VLAN isolation.

AFDX integrates redundancy in the transmission of messages, through the definition of two physical links for each end system. Each end-system gets two output Ethernet port, each one connected to a dedicated switch. It means that the network infrastructure is doubled, and each system is built with two Ethernet ports (A and B). So, each end-system is always connected to two identical switches, port A to switch A, and port B to switch B (see figure 3.4). Each message sent by the end-system is sent to both ports (to both switches, then) and the destination end-system is then responsible for getting them, eliminating the potential duplicate messages (each network link is doubled, so each message is sent twice, once per network copy) and, eventually, destroy redundant messages. To identify this redundance, each message is tagged with a bit indicating the network copy it belongs to.

The principle is simple: each end-system can be connected to each VLAN (up to a maximum if 128 per end-system) and each connection can be for transmission, or reception. The transmission of messages in each VLAN is done according to a multicast method: each reception-connected end-system will receive the emitted message, forwarded by the different switches.

Protocol description

As it has been said, AFDX is based on Ethernet 802.3 protocol. Basically, it needs to manage source and destination addresses and operates mainly at the first four layers of the ISO model (Physical, Link, Network, Transport). We propose here to focus on the precise definition and structure of AFDX according to each one of these layers.

Physical layer:(layer 1) Each R_x port (designed for reception) is linked to an input buffer. That allows

an incoming message to be stored before being transmitted to the switch, instead of being dropped out because of an overload. Then, when the transmission of the current message is finished, the switch can pick the next waiting message in the input buffer corresponding to the port.

Link layer:(layer 2) In terms of addressing, a MAC destination address (statically defined by the designer) is assigned to each VLAN. This MAC (48 bits size) is splitted in two parts [36]: The 32 first bits is a constant, identical for each end-system in the network, and the 16 other bits are the VLAN identifier. It means that an AFDX topology can contain, at the most, $2^{16} = 65536$ different VLANs.

3.2.4 AVB architecture

What is AVB?

AVB (Audio-Video Bridging) is a network protocol based on Ethernet [37]. It is issued from a global reflexion of the IEEE802.3 'Residential Ethernet Study Group' work in 2004. It was originally built as a low-cost improvement for COTS Ethernet in order to assure QoS in multimedia communications through Ethernet. Its internal mechanisms of traffic shaping allows us to manage various network traffics with high constraints of QoS and timeliness.

This protocol was originally conceived for multimedia flows management due to its potential to cover high needs in terms of QoS management, but an industrial implementation was proposed in [38].

AVB defines two classes of messages (*A*, *B*) and proposes a solution for traffic shaping of both transmissions of flows from these classes. In order to privilege class A and class B flows, the point is to base their transmission on credit level management. As long as the credit of a node is positive, we send class A messages. During idle phases (when credit is negative), we send class B messages if the credit of class B is positive or null. During both class A and class B transmission phases, the credit is respectively increasing or decreasing at pre-defined rates. It means that class A and class B messages transmission are mutually exclusive.

Finally, all flows which are not part of class A or B are sent with following a 802.1Q best effort policy (no traffic shaping): if there is a free time slot with negative credit or no class A/B messages waiting, we use this slot to send these best efforts messages.

Message classes

In terms of message management, AVB defines two classes of messages. Message transmission with AVB standards is based on the definition of different classes of messages, each one linked with a dedicated objective in terms of latency in the transmission and with a limited number of possible hops in the network. These classes are organized as follows:

- Class A messages: these messages are defined as the most important to send first in AVB network. infrastructure. They are defined as messages with high objectives of latency (2 ms of end-to-end delay), and a maximum of 7 hops along their path in the network. Usually, we consider class A messages in AVB as the most constrained one to privilege.

- Class B messages: Objectives of latency (depending on the network bandwidth) are slightly lower compared to class A messages (50 ms of latency). These class B messages are considered as global traffic to assure with specific constraints.

- Best effort (Class Z): Another class based on 802.1Q protocol is also available for best effort. These regroup all the other messages of the infrastructure, integrating not any particular constraint in terms of reliability or latency management.

AVB is a clock-synchronized network protocol designed as IEEE 802.1BA [39]. It was initially conceived as a protocol for RT media communication over Ethernet. Video transmission (cameras, streaming, ...), audio communication (VoIP), and other media communicaiton protocols are based on this standard to share data. AVB provides its own clock-synchronization protocol [40].

3.2.5 Time-Triggered Protocols

The time-triggered protocol defines time slots and allows only one node to transmit messages per time slot. For each node, the length of the interval can be configured by the system designer.

TDMA collision management

With this protocol, there will not be conccurence in message transmission between the different network nodes. Two nodes cannot send a message at the same time, so this avoids collisions. Time Division Multiple Access (TDMA) collision management protocol represents the fundamentals of time-triggered architectures [41].

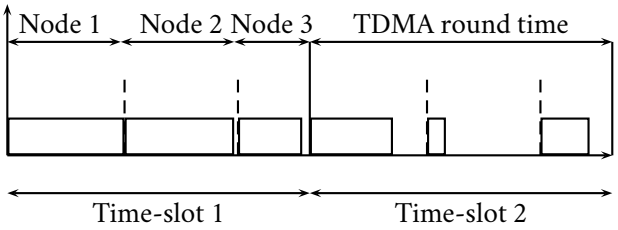


Figure 3.5: TDMA time slots

Time-triggered protocols are based on a mono-master/multi-slave architecture. There is one single node defined as the master, which is responsible for determining the duration of the transmission time interval for each slave node. In TDMA, determining these durations is based on a polling system, which was detailed in [42].

Each node must have the exact same value for each time slot and subslot bound. As a result, time-triggered protocols suppose that all the nodes in the topology are clock-synchronized. Depending on the implementation of TDMA, the infrastructure can rely on different clock-synchronization protocols.

Time Triggered Ethernet

TTEthernet [43], [44] is an industrial commercial-licenced implementation of the time-triggered protocol. It is mainly used in spaceship design and public-oriented real-time networks (personal vehicles,

public transport). TTEthernet is an implementation in Ethernet 802.1 of the TDMA collision management protocol. It provides the same time-triggered mechanism, based on splitting the time in discrete slots, and each of this slot subsplitted for guaranteeing it to a dedicated node.

The purpose of TTEthernet is to use the time-triggered slot splitting to allow all the nodes in the network to transmit important messages (mechanical control, radar, ...), safety messages (oxygen management, passengers protection, ...) and comfort messages (air conditioner, entertainment, ...) inside the same infrastructure. The TDMA protocol allows the network to answer to the isolation problems which are induced by such mixed transmissions.

Similarly to AFDX, TTEthernet integrates VLAN management. The TTEthernet implementation allows the messages to be isolated in their transmission and thus, this induces a reliable priority management during flows scheduling.

In TDMA, we suppose that each node is associated to a specific time slot for message emission. In TTEthernet, each network switch has its own dedicated time slot. In order to make this association between nodes, messages and time slots, TTEthernet proposes to define different classes of messages. During a time slot dedicated to a specific node, messages are organized according to classes of different importances. Implementing these classes and the TDMA structure applied to messages transmission assures the timeliness and safety different guarantees required by the network. We can represent TTEthernet messages accoring to 3 different classes(see figure 3.6).

- Time-triggered messages (TT): messages with the highest constraints. These messages must be transmitted with the highest accuracy. We can define static priorities and dedicated VLAN for TT messages in order to differentiate them in terms of priority. Each transmission of a TT message guarantees that, during the time slot, the source and destination nodes are entirely available specifically for this message transmission.

- Rate-constrained messages (RC): these messages have lower constraints in terms of performances and transmission delay, but still must be transmitted in a bounded time. The transmission of these messages relies on bandwidth allocation for each node (same as TT messages) but RC messages can suffer for waiting delays. Several RC messages can be sent to the same destination node and be queued before transmission. There is no lock on the destination port.

- Best-effort messages (BE): Classically following standard Ethernet, the transmission of these messages does not rely on any determinism nor maximum delay. These messages are treated with a lower priority than TT and RC messages and should be transmitted when possible, with no guarantee.

The figure 3.6 shows a simple example of message classes repartion among different time slots. Obviously, TT messages are transmitted first in the node, considered as the one with highest priority among all the rest of the node traffic.

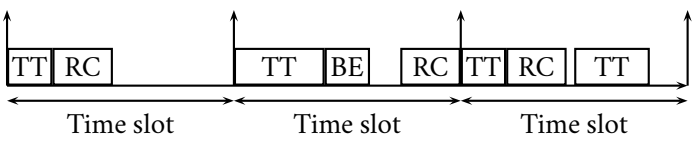


Figure 3.6: Time-triggered Ethernet messages

FTT-SE

Flexible Time Triggered Switched Ethernet (FTT-SE) is a time-triggered protocol designed for Ethernet networks. It was first introduced in [33], [45], and it is mostly used in industrial domains like automotive or avionics. Contrary to the TTEthernet, FTT-SE does not rely on specific protocol-compliant materials. It requires a clock-synchronized Ethernet network, but network devices do not have to be specifically dedicated to its implementation. It improves the genericity of the network, its reusability and decreases its cost. This allows us to integrate open time-triggered solutions even in public-oriented industrial products (personal vehicles, public transport, ...).

In terms of architecture, FTT-SE is a master-slave protocol: the master transmits a message from class TM (Trigger Message) to all the slaves. This message defines the duration and schedule of the next time slot. This allows the network nodes to propose a dynamic time-slot management and to provide different repartition of subslots to each node, depending on the network configuration and contraints. Each TM message indicates to all the slaves the beginning of a new time slot of duration d_{sync}.

Like we can see in figure 3.7, we can have a period between two TM messages which is longer than d_{sync}. The exceeding time can be used to transmit messages with lower importance classes during the created asynchronous time interval.

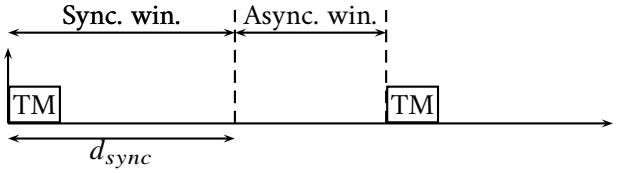


Figure 3.7: FTT-SE synchronous and asynchronous transmission

FTT-SE allows the transmission of different classes of network traffic:

- Synchronous messages, controlled and automatically triggered by the network, synchronized with TM messages durations (d_{sync}).
- Asynchronous messages, launched by external applications and transmitted with Best Effort policy.

Through synchronous and asynchronous messages classes, FTT-SE integrates the distinction between RT and non-RT traffic. It provides guarantees on the transmission time of RT messages. Through the management of more than 8 levels of priority (based on 802.1D [46]) and the integration of a QoS manager, it can assure QoS guarantees, even on non-RT traffic.

3.3 Clock synchronization with PTP protocol

In this work, we established a split between two main different network architectures: clock-synchronized architectures (TTEthernet, AVB) and non-synchronized architectures (AFDX, COTS Ethernet).

Mainly, there exist two major clock synchronization protocols which are considered as standards in networks and RT domains. These protocols are Network Time Protocol (NTP) [47] (for global uses) and

PTP[48] (for high constraints of accuracy). Each one of this protocol is based on the same hypothesis: one clock is defined as the time reference and the protocols assures the synchronization of other clocks to the reference.

In order to define the clock reference in the network, each protocol integrates different processes. These processes can be based on priority assignment, dynamic polling decision, or other solutions. More details about the clock reference definition were given in [49].

These protocols define the stratum of each clock (distance between the clock and its reference clock). This allows to define a relative clock accuracy for network with high number of nodes (NTP over Internet, for example). In our work, we rely on high accuracy requirements (magnitude μs) in terms of clock-synchronization. To do this, we focused on the PTP protocol.

3.3.1 Synchronization

PTP-ETE

PTP is a high-precision clock-synchronization protocol, used in RT and industrial networks. It offers precision of 100 ns [49]. In classical Ethernet context with 100 Mb/s (or 1Gb/s) of bandwidth, usually the WCAT has values between 0.05 and 100 μs. PTP is then a protocol of adequate precision for this context.

PTP protocol operates at the application layer of the OSI model (see figure 3.8). It means that, independently of the network latency, clock synchronization precision suffers from two different delays:

- Network delay, which represents the delay to transmit the PTP messages between master and slaves.
- Protocol delay, which corresponds to the software latency induced by all the layers of the OSI model. This delay needs to be taken into account in each side (master and clock) to compute the jitter between the two clocks.

The purpose of PTP is to synchronize the clocks of all slave nodes with the master clock. In order to do this, we use the protocol described below. Clock synchronization through PTP is based on the approach of synchronizing individually each slave directly with the master. Both master and slave clocks exchange a set of messages in order to synchronize themselves. Each one of this synchronization messages has its own size and transmission delay. We can detail these messages as:

- Announce message: used to indicate the clock configuration to other nodes in the network. This message is sent on PTP implementation in order to integrate the master-slave architecture and indicate which node is the current master.
- Synchronization message $Sync$: this message is used to call a slave to synchronize itself with the master. This message initiates a synchronization phase between the master and a slave clock. In terms of data, this message contains the emission time T_1 of the master clock, and different other clock properties used for configuration purposes.

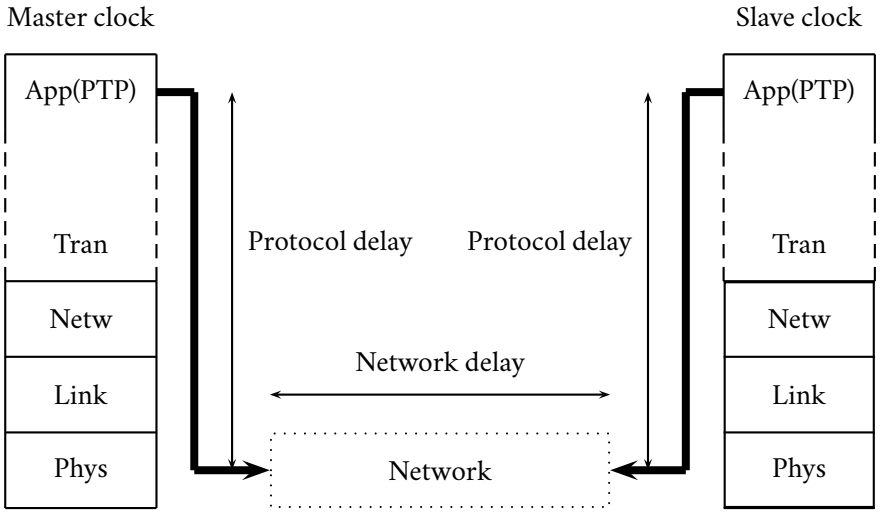


Figure 3.8: PTP clock jitter

According to the master-slave architecture, selecting the master (most accurate clock) in the network is done in PTP according to a specific algorithm called the Best Master-Clock Algorithm (BMCA). This algorithm is detailed in [50].

The synchronization phase of master and clock is detailed in figure 3.9. First, the master transmits a $Sync$ in order to call a slave to synchronize its clock with the master. Then, the master transmits to the slave the emission instant T_1 of $Sync$, through the F_{up} message. The $Sync$ message has a transmission time of t, and we suppose its clock jitter with the slave equal to δ.

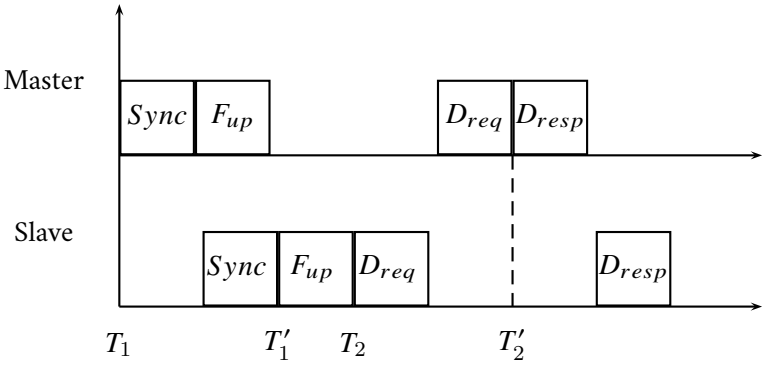


Figure 3.9: PTP end-to-end synchronization

- Follow up message F_{up}: this message is transmitted as a confirmation of the $Sync$ message. F_{up} is used to transmit the emission timestamp (T_1) of $Sync$ message from the master clock to its slave and to confirm the value of T_1 to the slave.

- Delay request message D_{req}: the slave answers to the master once the slave received a $Sync$ message. At the emission of $Delay_{req}$, the value T_1 and the value T_1' corresponding to the emission instant of $Delay_{req}$, are stored in the slave's internal memory.

- Delay response message D_{resp}:. The master answers to $Delay_{req}$. This $Delay_{resp}$ message contains the value of the date T_2 at which the master received the $Delay_{req}$ message.

As soon as it got the F_{up} message, the slave sends a D_{req} message to the master, emitted at date T_2. The value of T_2 is stored in the slave's memory. Once the master gets the D_{req} message, it stores its reception instant T_2', and sends it back to the slave with a D_{resp} message. At the reception of the D_{resp} message, the slave gets the following values: T_1, T_1', T_2, T_2'.

The PTP clock-synchronization process is based on the assumption that, between two different nodes, the transmission time of a message is constant (supposing no additional delay due to waiting queues). It

means that both $Sync$ and D_{req} messages are supposed to be transmitted in the same time t. Based on this hypothesis, we obtain:

$$\begin{aligned} T'_1 - T_1 &= t + \delta \\ T'_2 - T_2 &= t - \delta \\ \delta &= \frac{T'_1 + T2 - T_1 - T'_2}{2} \end{aligned} \tag{3.1}$$

According to the computation of δ, we can correct the clock value of the slave and eliminate the existing jitter δ existing between the two clocks. Finally, the slave has to adjust its internal clock by correcting it with δ value.

When synchronizing two different clocks which are separated by, at least, one intermediate node, this intermediate node will be defined as a transparent clock for PTP. It means that this transparent clock will just have the role to forward the synchronization messages from master to slave, and reversely.

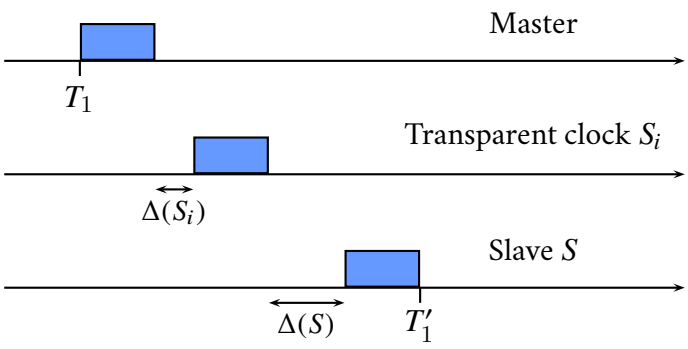


Figure 3.10: PTP-ETE transparent clocks

In PTP-End To End (PTP-ETE), the synchronization of two clocks separated by, at least, one intermediate node, induces addition clock jitter to correct (see figure 3.10). For each transparent clock TC_i in an intermediate switch S_i, we add to the synchronization delay the transmission time (between the current switch and the previous one) which is denoted as $\delta(TC_i)$ and the clock jitter induced by the current node, denoted as $\Delta(S_i)$.

Basically, computing the value of δ and of different transmission times T_1, T'_1, T_2 and T'_2 is based on the assumption that the transmission delay between master and slave is induced by two different sources: the transmission time induced in each intermediate transparent clock, and the clock jitter introduced by each intermediate transparent clock. Considering an number n of transparent clocks in intermediate nodes, we can compute the following expression:

$$T'_1 = T_1 + \sum_{i \in [0;n]} (\delta(TC_i) + \Delta_{S_i}) \tag{3.2}$$

PTP-P2P

In PTP-ETE, when there is more than one physical link between the slave and the master, we need to forward the synchronization messages through the intermediate nodes. These nodes are considered as transparent in the synchronization process, forwarding the synchronization message between the master and the slave. Even if PTP-ETE can manage the clock synchronization between two nodes separated from many intermediate nodes, the successive forwarding through intermediate nodes can induce a lack of accuracy in the synchronization process. By adding successive different jitters, we multiply the number of approximations when computing the transmission time of a message.

PTP-ETE is based on the assumption that the value of δ is constant for all transmissions, which could appear to be false in case of a high number of successive transitions. Increasing the distance and the number of intermediate nodes between master and slave make the hypothesis of constant transmission delay unreliable. In order to answer to this problem, PTP-Peer To Peer (PTP-P2P) proposes an alternative way to synchronize clocks in the network.

PTP-P2P is an alternative solution for clock synchronization with PTP. Contrary to the slave master architecture which defines one master clock to the whole network to synchronize to, PTP-P2P is based on a peer-to-peer approach. Each synchronization of a clock is done only with its closest neighbours. It allows us to limit the transmission delay computation to two directly connected nodes. PTP-P2P does no longer rely on the different evaluations of timestamps computed by the master and the slave, but directly on the transmission time needed to transmit the message from peer to peer.

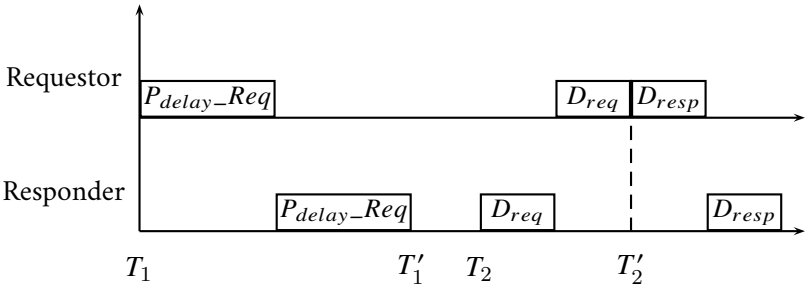


Figure 3.11: PTP peer-to-peer synchronization

The synchronization process of PTP-P2P is described in figure 3.11. Each peer transmits a $P_{delay\_}Req$ to the closest neighbor, which answers with a $P_{delay\_}Resp$ containing its current timestamp.

Eventually, we apply the same timestamp comparison as for PTP-ETE to compute clock jitter between master and slave. In case of non-deterministic networks with an automatic route computation (spanning tree algorithm), PTP-P2P has been proven to be more reliable [48]. The transmission delay between master and slave has not necessarily to be the same in both ways to allow a clock synchronization.

As a conclusion, PTP-ETE implies only the master and the slave to be PTP-compliant, whereas PTP-P2P implies all the encountered nodes to be able to identify a PTP message. It means that PTP-P2P implies all the switches in the network to be able to manage clock synchronization. In RT networks, we assume that we are working mainly on closed topologies with static-defined paths (see section 3.1). PTP-P2P could seem more convenient to our purposes but, as it implies strong constraints on the infrastructure, PTP-ETE appears as a costless solution.

3.3.2 Frames

PTP frames were detailed in the PTP specification [48] and some other works [49]. We do not want to make an exhaustive description here, but only explain the fundamentals we need in our work in terms of frame modelling and clock-synchronization integration.

All PTP frames are based on a common frame structure composed of a header (34 bytes, detailed in [48]). a body and an optional suffix. In order to fully understand the structure of PTP frames, we detail here the model of each type of PTP message.

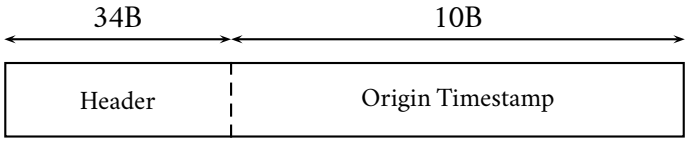


Figure 3.12: PTP body - F_{up}, $Sync$ and D_{rep} messages

Each PTP message has a different body structure, depending on its role. It means that each PTP message has a dedicated size in bytes.

In terms of frame model, $Sync$, $Follow_{up}$, and D_{req} messages are built on the same frame model, containing the timestamp of the sender (see figure 3.12).

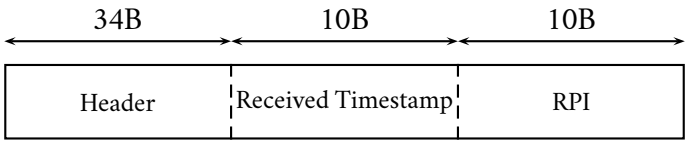


Figure 3.13: PTP body - $Delay_{resp}$ message

The D_{resp} message contains a timestamp, but also a Requesting Port Identifier (RPI). The RPI identifier is used to identify the emission port of the message from the master node (see figure 3.13)

Following the same structure, we detail here two frames of PTP-P2P mode. We detail here the frames for $Pdelay_{req}$, $Pdelay_{resp}F_{up}$ (see figure 3.14) and $Pdelay_{resp}$ (see figure 3.15) messages.

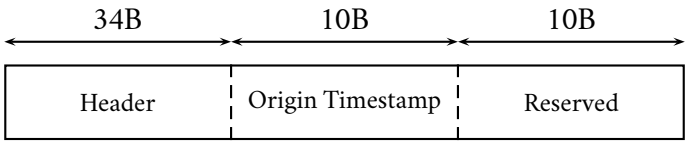


Figure 3.14: PTP body - $PDelay_{req}$ frame

This frame from $PDelay_{req}$ is composed of 10 bytes of reserved field. This field is used in order to adjust the message size to be the same as $PDelay_{resp}$ size (padding).

The PTP-P2P protocol relies on a symmetric exchange of messages, implying $PDelay_{req}$ and $PDelay_{resp}$ to be of the same size.

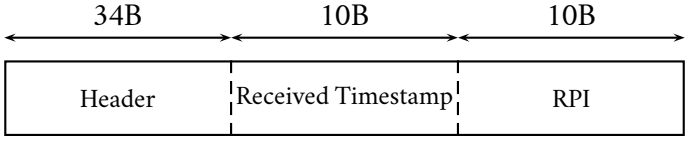


Figure 3.15: PTP body - PD_{resp} and $PD_{resp}\_F_{up}$ frames

Apart from these padding bytes, we can affirm that $PDelay_{req}$ has a very comparable structure to $Delay_{req}$. Similarly, the $PDelay_{resp}$ frame structure is the same as for $Delay_{resp}$.

3.4 Conclusion

In this chapter, we presented the fundamentals of RT networks and especially the different RT Ethernet infrastructures. Some of these infrastructures rely on clock-synchronization, which represents an additional constraint in terms of costs and devices configuration. Depending on the context (avionics, automotive, public transports), this synchronization can be seen as a global parameter to implement everywhere in the network, or just in a set of precisely selected nodes.

Our purpose in the following chapters would be to analyze and propose solutions to guarantee the timeliness and safety of these clock-synchronized and not clock-synchronized architectures, particularly in terms of transmission delays. That is why we propose, in the next chapter, to detail the existing transmission delay computation methods.

Chapter 4

End-to-end transmission delay computation

"L'homme qui déplace les montagnes commence par les petites pierres"

"The man who moves a mountain begins by carrying away small stones."

– Confucius [51]

Contents

4.1 Introduction

How to propose deterministic methods to compute worst case transmission delay of a message inside a network?. We present here the existing methods to answer to this question and their limits. In this chapter, we detail three major computation methods applicable to RT networks: the holistic method, the network calculus method and the Trajectory Approach. The purpose of each is to compute the worst case response time R_i of a message m, issued from a flow v_i.

4.2 Other computation methods

4.2.1 The holistic method

The holistic delay computation method [52], [53] is considered as the "historical" method. It can be considered as a default solution for delay computation in processor-oriented context and RT networks. It is based on a generic modelling of a network: this solution has been extracted from RT processor context [54]. The delay computation through the holistic method has been first defined for distributed architectures, and then extended to network context.

Computing transmission delay with the holistic approach consists applying the computation method to a specific node, then extending the computation progressively to all nodes of the network by following an iterative process. At each encountered node along a message path, we compute the smallest and the highest release time of the message. These time instants depend on the delay induced by network traffic.

The holistic approach considers that, at each node, a message can be delayed by a specific jitter due to other messages. In order to compute the worst case end-to-end transmission delay of a message, the holistic approach proposes to bound this jitter and to make a worst case evaluation of it on each node. Then, we can deduce the transmission delay of a message as the combination of all the computed jitters.

Limits

The holistic approach considers that each message that can pass through a node and delay the transmission of m will do it. This is safe but it can lead to a very pessimistic evaluation of the delay. The holistic approach takes into account all potential cases of network and messages configuration when computing transmission delay, which implies to take into consideration physically impossible situations. This can integrate pessimism in the method.

4.2.2 Network calculus

Concept

Network Calculus has been introduced [55] as a delay computation solution in RT networks. It has been presented [56] as a network performances computation tool. Network Calculus is a set of mathematical

tools used to verify performances and assure guarantee satisfaction in network topologies. A detailed application of the method for avionic networks can be found in [57]. Computing worst case end-to-end transmission delay with Network Calculus has been treated in [58].

In Network Calculus, we consider each node in a network topology as a computing unit, which gets data as input at a specific rate and produces data as output at another rate. The model of network calculus is not based on the distinction of flows like for Holistic Approach, but directly on amounts of produced data for each node. Considering this approach, we can compute the delay induced by a message in all encountered node along its path. We can provide an expression of the worst case end-to-end transmission delay of a message in the network.

Limits

Network Calculus presents several problems. First, it is complicated to model and implement: its applicability to all network contexts makes it difficult to adapt to concrete cases.[57] showed the complexity to represent and compute the bounds of transmission delay with Network Calculus. More specifically, the Network Calculus computation time tend to strongly increase when applying the method to nodes interconnected through various topologies [59].

Network Calculus is costly to use in terms of time and ressources. As a conclusion, we want to introduce an end-to-end delay computation method which has been defined specifically for RT industrial networks and costs less to use. This method is called the Trajectory Approach and is detailed below.

4.3 Trajectory approach

4.3.1 Presentation

The Trajectory Approach was presented in [60]. It is an end-to-end transmission delay computation approach which proposes a delay analysis based on the trajectory of a flow instead of each node in the trajectory (unlike Holistic approach and Network Calculus). The Trajectory Approach is based on its own network modelling. This modelling consists in representing a network as a set of interconnected nodes, and representing the data as flows following statically-defined paths.

[52] introduced the Trajectory Approach as an alternative of the holistic approach in the list of delay computation methods. The principle is as follows: we model a network as a set of flows (see figure 4.1), and the worst case transmission delay of a message (from a given flow) is computed according to each node encountered along its path. The computation of the worst case transmission delay of a message does not anymore consider all the nodes in the network but only the node in the path of a flow.

This approach solves two issues: First, we do not anymore consider physically impossible situations taking into account irrelevant nodes: we only focus on flows likely to impact the delay computation . Secondly, as we consider less nodes than the previous methods, it eases the computation of the final expression of delay computation.

The Trajectory Approach is a retro-processing method: we do not analyze the delay starting from

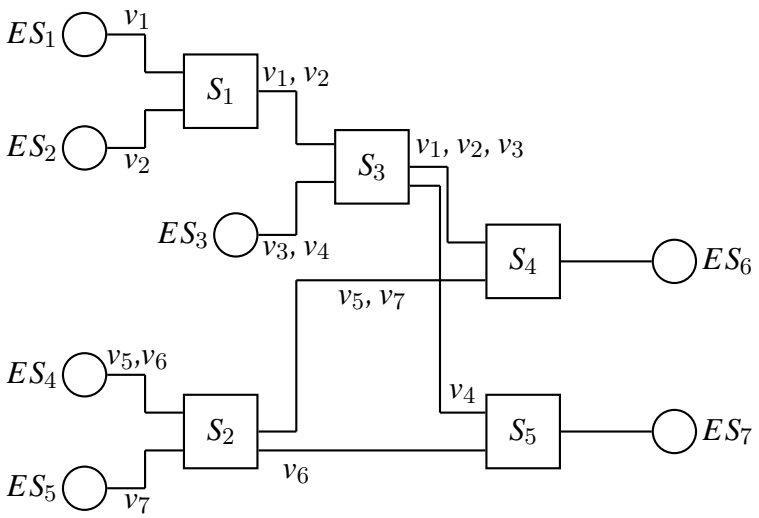


Figure 4.1: Network flows modelling

Modelling a network with the Trajectory Approach relies on the flows definition. Each flow is defined with its source and destination nodes in the network (end-systems). The method is applicable to various topologies, as long as the network flow paths do not integrate loops: they cannot cross twice the same node.

the first node in the path of a message. On the contrary, we start from the flow destination node, and we add one by one all the possible delays induced by network traffic in the other nodes, until we reach the source node of the flow.

The Trajectory Approach is built for periodic and sporadic flows. A flow v_i is characterized by the parameters $\{\vec{\mathcal{P}}_i, C_i, T_i\}$, repespectively the flow path, WCAT and period (or minimum inter-arrival time for sporadic flows).

The Trajectory Approach relies on the notion of busy period. A busy period (see figure 4.2) in a node is a time interval during which there is no idle time.

From its start to its end, there is a continuous transmission of messages. A busy period is a period of time during which a node is entirely busy transmitting successive flows of data.

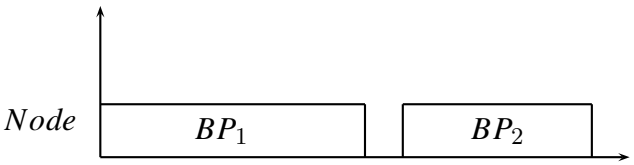


Figure 4.2: Busy periods

4.3.2 Model with FIFO

The transmission of a message m from a flow v_i can be delayed by different sources. In Trajectory Approach like for all other delay computation methods, this switching latency is considered as an upper-bounded constant for all the links in the networks. It is denoted as sl.

When the message m arrives in a given node, the worst case assumption implies considering that it will be delayed by all messages with a higher or equivalent priority. We consider that all messages with a higher or equal priority as m will be transmitted before m.

All the messages transmitted before m represent a delay which needs to be bounded. This delay is expressed, for each delaying flow v_j, as the result of the flow WCAT(C_j) and the number of transmitted messages from v_j which will have an impact on the transmission delay of message m.

During the focused time interval, each flow with a higher than or equal priority to v_i will produce a certain number of messages likely to delay the transmission of m. The purpose of the Trajectory Approach is to compute the number of transmitted messages during this time interval, for each flow. To do so, we introduce the following concepts:

- FT^n_{min} is the smallest WCAT among the messages transmitted during the busy period on node n. It is computed by: $FT^n_{min} = \min_{j\in\{1,2,...,n\}, n\in\vec{\mathcal{P}}_j}(C_j)$.
- $S^n_{max_i}$ and $S^n_{min_j}$ are the maximum and the minimum delays experienced by flow v_i from its source node $first_i$ to the output port o of node n.
- M^n_i is considered as the earliest arrival time of the first message that will delay flow v_i on the output port o of node n. It is computed by: $M^n_i = \sum_{j=first_i}^{n-1}(FT^j_{min} + sl)$.
- $\mathcal{V}$ is the set of flows likely to be transmitted in the network.
- n_f is the number of flows in the network $\mathcal{N}$. We have $|\mathcal{V}| = n_f$.

As we are working with periodic and sporadic flows, the delay induced by each flow can be computed by knowing the time interval when all the transmitted messages are likely to delay m. For messages from a flow v_j likely to delay m in node k, the time interval corresponds to the following value $A_{i,j} = S^k_{max_i} - S^k_{min_j} - M^k_i + S^k_{max_j}$, with: with k the first encountered node of v_i and v_j along their respective path. The computation of this value has been detailed in [61].

Knowing the length of the time interval, we can compute the number of transmitted messages from flow v_j with $(1 + \lfloor \frac{t+A_{i,j}}{T_j} \rfloor)$. This value allows us to determine, for each flow v_j encountered by v_i, the potential delay induced by v_j on the transmission of message m. As a conclusion, we can express the impact of higher and equal priority flows on v_i end-to-end transmission delay with the following expression:

$$\sum_{\substack{j\in\{1,2,...,n_f\} \\ \vec{\mathcal{P}}_i\cap\vec{\mathcal{P}}_j\neq\emptyset}} \left(1 + \left\lfloor \frac{t + A_{i,j}}{T_j} \right\rfloor\right)^+ * C_j \tag{4.1}$$

Added to the delay represented by the network traffic, we have to consider the WCAT of message m, which is added once per encountered node along $\vec{\mathcal{P}}_i$. But, in each node, the length of the different messages can occur to represent an additional delay. To represent this, we have to consider the longest message in each encountered node. We obtain the following expression :

$$\sum_{k\in\vec{\mathcal{P}}_i} \left(\max_{\substack{j\in\{1,2,...,n\} \\ k\in\vec{\mathcal{P}}_j}} (C_j) \right) \tag{4.2}$$

4.3.3 Non-preemptive delay

When m arrives in a node, there could be a message which is being already currently transmitted. No matter the priority of this message, its transmission cannot be stopped, which could induce an additional delay, called the non-preemptive delay. This delay is integrated in the Trajectory Approach computation.

We consider message 2 with a lower priority than message 1. Nevertheless, when 1 arrives in node S_1, 2 is being already currently transmitted. The delay induced by the end of the transmission of 2 is an illustration of the non-preemptive delay. NP^1_{S1} is the non-preemptive delay applied to the transmission of message 1 on node S_1.

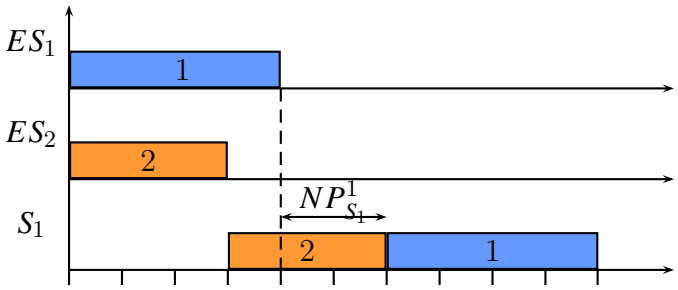


Figure 4.3: Non-preeemptive effect

For a worst case analysis, we consider the longest possible non-preemptive delay in the node. In FIFO context, there is no priority management in this context. But, messages from other VLAN, inducing in each node the message with the longest WCAT as a non-preemptive effect. As a conclusion, we can express the non-preemptive delay μ^i_n in node n of flow v_i with the following expression:

$$\mu^i_n = \max_{\vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset} (C_j) \tag{4.3}$$

In order to represent the whole non-preemptive delay induced in each node of the network, we add the potential non-preemptive delay induced to the transmission delay of m, in all nodes from $\vec{\mathcal{P}}_i$ (see expression 4.8). We obtain:

$$\sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu^i_k = \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \left(\max_{k \in \vec{\mathcal{P}}_j} (C_j) \right) \tag{4.4}$$

The non-preemptive delay computation in the Trajectory Approach is detailed in [62], [63].

4.3.4 Global delay computation

In our application of the Trajectory Approach, we consider FIFO scheduling of messages : the priority in FIFO in indexed on a message arrival time. Considering the previously computed delays (due to higher

priority traffic and non-preemptive delay) we can establish the delay induced by network traffic on the transmission of a message m from flow v_i. Computing worst case delay transmission with the Trajectory Approach consists in computing the delay experienced by a message m along the path of the flow v_i m belongs to.

The end-to-end transmission delay of a message could be summarized by the combination of expressions 4.7 and 4.9. But this delay is not complete. It must take into account the switching latency experienced by m. This latency is upper-bounded by sl for each transmission link encountered along the path of v_i. We can summarize the expression of this switching latency as $|\vec{\mathcal{P}}_i| * sl$.

Combining these expressions, we can express the latest starting time of m from the last node $last_i$ in the path of v_i. This delay is illustrated in figure 4.4. We note this delay as $W_{i,t}^{last_i}$, considering that m was emitted at time instant t in its source node.

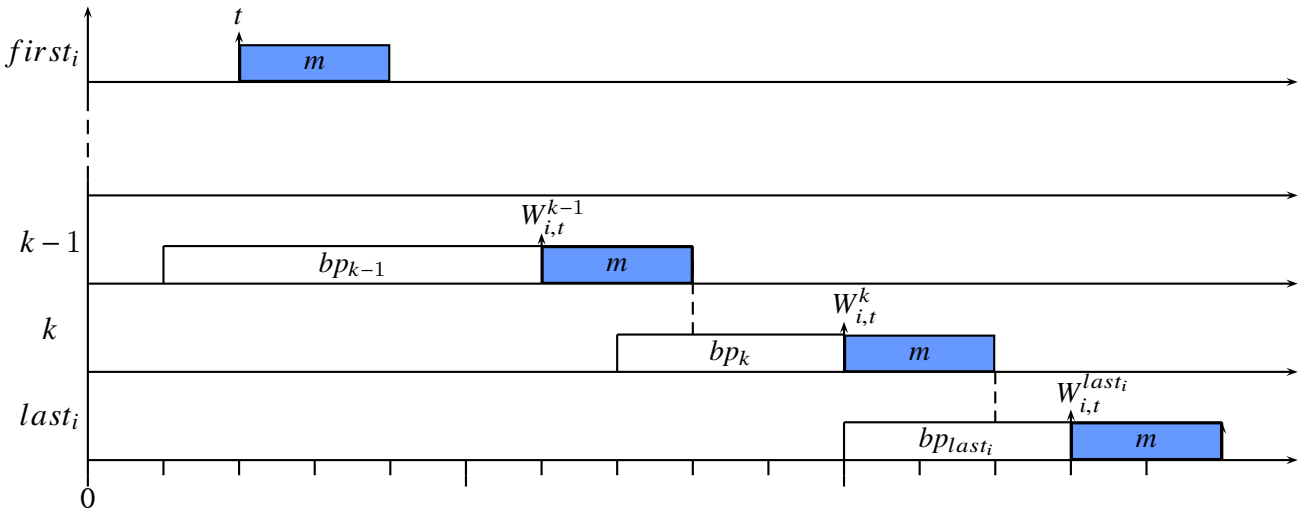


Figure 4.4: Trajectory approach details

In order to compute the response time R_i of m from v_i, we first compute the latest starting time of m from the last node in its path (denoted as $last_i$). This delay is denoted as $W_{i,t}^{last_i}$ and relies on the release time t of m from its source node $first_i$. Considering the previous work, we can express $W_{i,t}^{last_i}$ with the following expression:

$$
\begin{aligned}
W_{i,t}^{last_i} = & \sum_{\substack{j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(1 + \left\lfloor \frac{t + A_{i,j}}{T_j} \right\rfloor\right)^+ * C_j \\
& + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \left(\max_{\substack{j \in \{1,2,\ldots,n\} \\ k \in \vec{\mathcal{P}}_j}} (C_j) \right) \\
& + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu_k^i \\
& + (|\vec{\mathcal{P}}_i| - 1) * sl \\
& - C_i
\end{aligned}
\tag{4.5}
$$

When computing the end-to-end transmission delay, we often use $W_{i,t}^{last_i}$ as a reference. In fact, starting from the last node in its path to its destination, m will not suffer from additional delay due to network traffic. It means that the difference between the response time $R_i(t)$ of m and $W_{i,t}^{last_i}$ only depends on t. We note that $R_i(t) = W_{i,t}^{last_i} - t + C_i$. The complete end-to-end transmission time of m can be computed by considering the worst case value of $R_i(t)$. This delay is denoted as R_i and is represented as follows:

$$
R_i = \max_{t \geq 0}(R_i(t)) = \max_{t \geq 0} \left\{ W_{i,t}^{last_i} - t + C_i \right\} \tag{4.6}
$$

4.3.5 Serialization effect

Basically, we assume that competing messages from different flows can arrive at the same input port of a node at the same time. In practical situations, competitive messages sharing the same link have to be serialized before being transmitted. [64] showed that minimizing the serialization by 0 was a source of pessimism in the Trajectory Approach. We detail here the computation of the serialization effect.

Martin et al. [61] ignored the serialization of frame sharing the same links (due to flow representation) in their approach. They minimized this delay by 0. In more recent works, Bauer et al. [65] showed that this minimization induced a strong pessimism in the approach. They proposed a new bound to the serialization term, which reduced this pessimism.

There is an induced serialization delay at each input link of a node h. All the flows connected as input of h are all connected to a specific input port of h, denoted as $IP_{j_h}^h$, with j_h the number of the input port. Each competing message connected to an input port IP_j^h comes from another output port OP_{h-1}, with $h-1$ the node before in the path $\vec{\mathcal{P}}_j$. As a conclusion, OP_{h-1} is a different output port for each flow.

Each message transits through a link between OP_{h-1} (the previous node) and OP_h (the output port

of node h) via the input port $IP^h_{j_h}$. Then, on the output queue, each message is selected and transmitted to its destination, according to the scheduling policy of node h (FIFO, fixed priority, etc...).

In order to explain the serialization effect, we introduce the respective terms FT^q_{min}, FT^q_{max} , which are the minimum and maximum WCAT present in the busy period of output port OP_h, and $FT^{IP^h_j}_{max}$, which is the maximum WCAT from the busy period in input port IP^h_j.

As shown in figure 4.5, competitive arrival of successive flows from different input ports and transiting to the same output port can represent an additional delay in the transmission of a message m from flow v_i. This phenomemnon is called the serialization effect and is represented, in each node h for each flow v_i, with the term $\Delta^h_i(t)$ (where t is the emission date of message m from v_i at its source node).

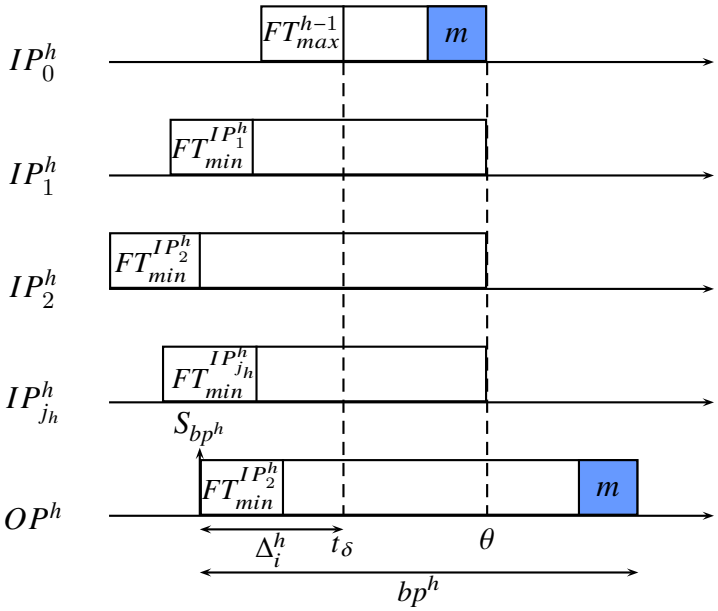


Figure 4.5: Serialization effect

Due to serialization, messages are likely to arrive in output port OP_h before the date t_δ. t_δ represents the arrival date of the first message from the busy period of the input port IP^h_0. All messages arriving in OP_h before date t_δ are not likely to delay m. These messages have to be substracted from the Trajectory Approach expression. This optimization of the Trajectory Approach was detailed in [65].

According to figure 4.5, the serialization effect Δ^h_i in node h is the delay between the beginning of bp^h and the arrival of the first message (m) from IP^h_0. As we search for the worst case transmission delay of m, we want to minimize the value of Δ^h_i. According to [65], the smallest value of Δ^h_i is computed when:

- The first message of bp^h is the smallest message of bp^{h-1}.
- The first message of each input port IP^h_j has the longest WCAT in the busy period $bp^{IP^h_j}$.

Both these conditions are illustrated in figure 4.5. In order to compute the serialization effect, we have to consider the potential effect induced in each input port, of each node. In each input port IP^h_n, the number of generated messages are represented by the number of flows v_j transitting through this port. This is indicated by:

$$\sum_{v_j \in IP^h_x} \left(\left\lfloor 1 + \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+ * C_j \qquad 4.7$$

Each flow from each input port can generate a potential non-preemptive effect, which has to be substracted from the serialization term. As a conclusion, in an output port IP_h, the flows transitting through this port are responsible of the following serialization effect:

$$\sum_{v_j \in IP_x^h} \left(\left\lfloor 1 + \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+ * C_j - \max_{v_j \in IP_x^h} (C_j) \tag{4.8}$$

The expression of the serialization effect is computed on each input port. Then, we consider the maximum delay among all input ports. We obtain:

$$\Delta_{i,t}^h = \max_{x \in \{1,2,\ldots,j_h\}} \left\{ \sum_{v_j \in IP_x^h} \left(\left\lfloor 1 + \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+ * C_j - \max_{v_j \in IP_x^h} (C_j) \right\} \tag{4.9}$$

Eventually, we substract the delay induced by messages from the same input port as m. These messages delay the transmission of m. Considering this, we obtain the following expression of Δ_i^h:

$$\begin{aligned} \Delta_{i,t}^h = & \max_{x \in \{1,2,\ldots,j_h\}} \left\{ \sum_{v_j \in IP_x^h} \left(\left\lfloor 1 + \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+ * C_j - \max_{v_j \in IP_x^h} (C_j) \right\} \\ & - \sum_{v_j \in IP_0^h} \left(\left\lfloor 1 + \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+ * C_j - \min_{v_j \in IP_0^h} (C_j) \end{aligned} \tag{4.10}$$

Considering this serialization term for each flow v_i and each node h in the network $\mathcal{N}$, we obtain the corrected expression of the latest response time computed by the Trajectory Approach:

$$
\begin{aligned}
W_{i,t}^{last_i} = & \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(1 + \left\lfloor \frac{t + A_{i,j}}{T_j} \right\rfloor\right)^+ * C_j \\
& + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \left(\max_{\substack{j \in \{1,2,\ldots,n\} \\ k \in \vec{\mathcal{P}}_j}} (C_j) \right) \\
& + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu_k^i \\
& + (|\vec{\mathcal{P}}_i| - 1) * sl \\
& - \sum_{k \in \vec{\mathcal{P}}_i \setminus first_i} (\Delta_{i,t}^k) \\
& - C_i
\end{aligned}
\tag{4.11}
$$

4.4 Optimism in the Trajectory Approach

In recent works [62], [66], it has been shown that there exist corner cases where the delay computation proposed by the Trajectory Approach appeared to be optimistic compared to real cases. This means that the Trajectory Approach in its current form can be considered as a non reliable method for end-to-end delay computation. Based on the work presented in [63], we want to expose the sources of this optimism and to propose a correction to the expression of the Trajectory Approach.

4.4.1 Example

We illustrate the optimism problem through an example. We consider the network topology and set of flows presented in figure 4.6. This network is composed of a set of 8 end-systems $\{ES_1, ES_2, \ldots, ES_8\}$ and 3 switches $\{S_1, S_2, S_3\}$ with 7 flows $\{v_1, v_2, \ldots, v_7\}$ transmitted through the network. We want to compute the end-to-end transmission time of a message m from flow v_1. Its path $\vec{\mathcal{P}}_1$ is equal to $\{ES_1, S_3, ES_8\}$ in the topology (see figure 4.6).

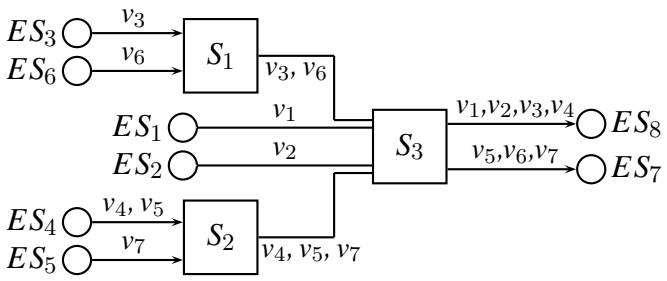


Figure 4.6: Illustrative topology for the Trajectory Approach

We suppose the switching latency $sl = 0\mu s$. We suppose that we are working with periodic and sporadic flows. The parameters of the different flows are given in the table below:

Flow	v_1	v_2	v_3	v_4
C_i (μs)	40	20	20	40
T_i (μs)	4000	4000	60	120

Flow	v_5	v_6	v_7
C_i (μs)	30	40	50
T_i (μs)	4000	4000	4000

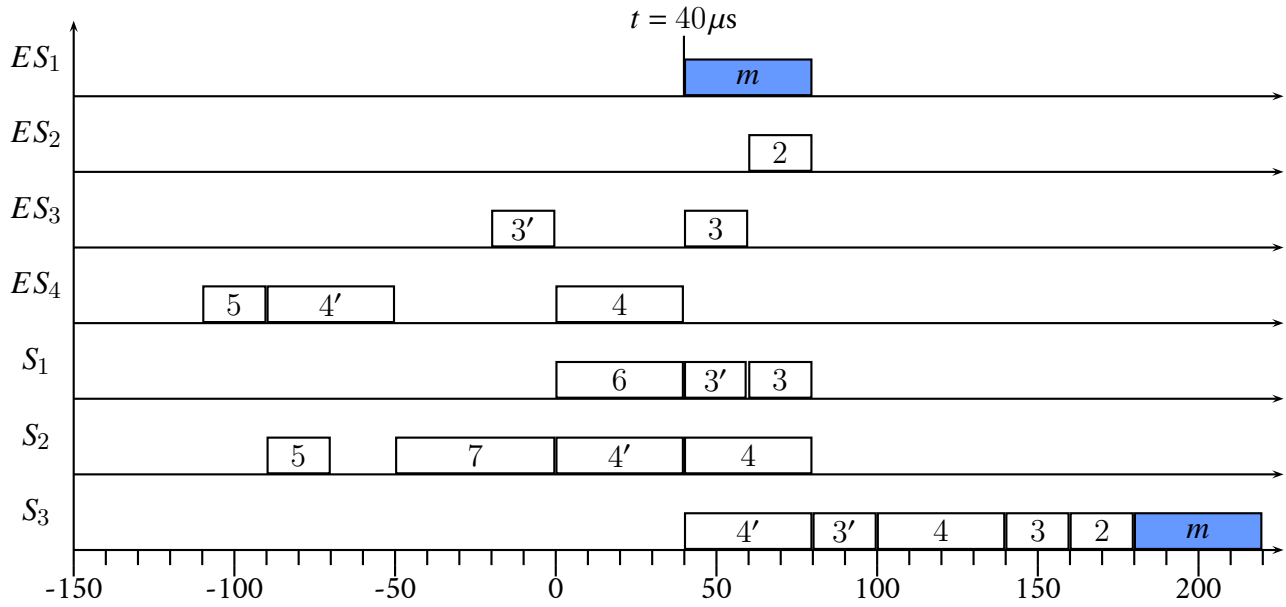


Figure 4.7: Transmission time of message m from v_1

The worst case scheduling scenario of m is represented in figure 4.7. We compute the latest starting time of m as the delay when m is transmitted from S_3 to its final destination ($W_{i,t}^{S_3}$). The worst end-to-end transmission delay of m is the duration between t and the arrival insant of m in ES_8 (destination node). We compute $R_1 = \max_{t \geq 0}(R_1(t)) = R_1(40) = 180\mu s$.

If we apply the expression 4.11 of the Trajectory Approach, we obtain $R_1 = \max_{t \geq 0}(W_{1,t}^{S_3} - t + C_1)$. We compute the different values:

$A_{1,2}$	$A_{1,3}$	$A_{1,4}$
0	40	80

In order to compute R_1 with the Trajectory Approach, we compute the different values of $A_{i,j}$ for the competing flows v_2,v_3,v_4.

We obtains the value $R_1 = 140 - 0 + 20 = 160\mu s$ (the detail of the computation can be found in [62]). This result is $20\mu s$ below the real transmission time. This leads to the following conclusion: in that case, the Trajectory Approach led to an optimistic result, representing an error margin of nearly 10 %. We want to explain the source of this optimism, and then to propose a correction to it.

4.4.2 Problem

In the work done in [63], we showed that the Trajectory Approach considers that message transmissions during the time interval $[M_i^{last_i}; M_i^{last_i} + t]$ are not likely to delay the transmission of message m.
When computing the transmission delay with the Trajectory Approach expression on the scenario described in figure 4.9,

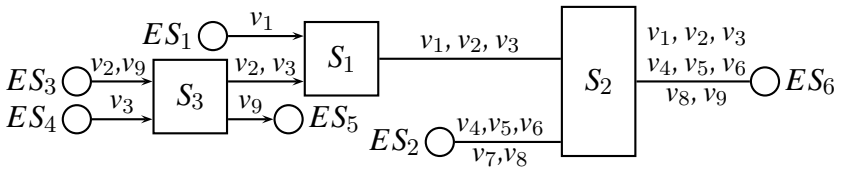


Figure 4.8: Topology for the characterization of the optimism due to serialization effect

Flow	C_i (μs)	T_i (μs)
v_1	40	4000
v_2	40	120
v_3	40	4000
v_4	40	320
v_5	40	4000
v_6	40	4000
v_7	40	4000

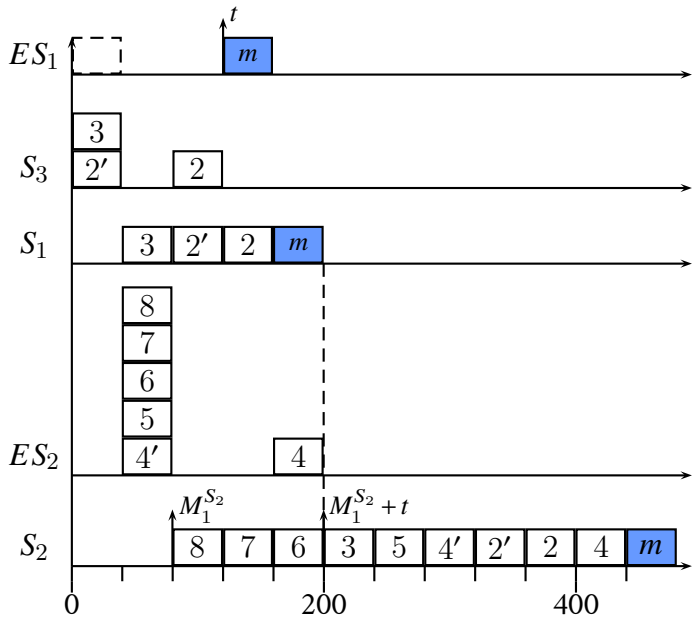


Figure 4.9: Serialization effect details

Considering these parameters, we obtain (with the classical approach) the results detailed in figure 4.9. The Trajectory Approach considers that different messages from different flows can arrive at the same time and delay m. On the contrary, it does not take into account the minimum temporal time between two consecutive messages of the same flow (contrary to the classical approach).
Combining this, it can lead to the point that all messages transmitted during the interval $[M_i^{last_i}; M_i^{last_i} + t]$ are substracted twice from the Trajectory Approach delay computation expression: once because the serialization term overlaps the time interval $[M_i^{last_i}; M_i^{last_i} + t]$, and once because that classical approach already substracts these messages. This double substraction leads to an optimistic result.

4.4.3 Correction of the serialization effect

As shown, the optimism in the Trajectory Approach comes from a mis-evaluation of the serialization term bound proposed in [65]. According to the classical approach, all messages likely to delay m are transmitted in node k after M_i^k. However, due to the serialization effect, some messages are transmitted before M_i^k and their impact on the end-to-end transmission delay of m should be substracted from the serialization delay. As a result, there is an overlap between the two intervals.

Some messages transmitted during the interval $[M_i^{last_i} + t; W_i^{last_i}]$ are likely to be substracted twice from the final expression. Our solution to the presented problem is to exclude this overlap from the serialization term. We proved in [63] that the overlapped time interval duration was equal to the smallest value between Δ_i^h and t. the serialization term can be corrected as follows:

$$\left(\sum_{k \in \vec{\mathcal{P}}_i \setminus first_i} (\Delta_{i,t}^k) - t \right)^+ \tag{4.12}$$

As a conclusion, we propose the following expression to compute the end-to-end last response time of a message m from flow v_i:

$$\begin{aligned} W_{i,t}^{last_i} = & \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(1 + \left\lfloor \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+ * C_j \\ & + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \left(\max_{\substack{j \in \{1,2,\ldots,n\} \\ k \in \vec{\mathcal{P}}_j}} (C_j) \right) \\ & + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu_k^i \\ & + (|\vec{\mathcal{P}}_i| - 1) * sl \\ & - \left(\sum_{k \in \vec{\mathcal{P}}_i \setminus first_i} (\Delta_{i,t}^k) - t \right)^+ \\ & - C_i \end{aligned} \tag{4.13}$$

4.5 Notations

- $\vec{\mathcal{P}}_i$ is the path of the flow v_i.
- C_i is the WCAT of the flow v_i.
- T_i is the period (or minimum inter-arrival time) of the flow v_i.
- v_i is a network flow, characterized by the properties $\{\vec{\mathcal{P}}_i, C_i, T_i\}$.
- $first_i$ and $last_i$ are respectively the first and last node of the path $\vec{\mathcal{P}}_i$ from flow v_i.
- $first_{i,j}$ is the first common node in the paths $\vec{\mathcal{P}}_i$ and $\vec{\mathcal{P}}_j$ from flows v_i and v_j.

- μ_i^k represents the non-preemptive effect induced by messages from other VLAN and likely to delay message from flow v_i in node k.
- $\left(\sum_{k \in \vec{\mathcal{P}}_i \setminus first_i} (\Delta_{i,t}^k) - t \right)^+$ is the serialization effect of messages transmission delay, introduced in [63] and detailed chapter 4.
- $W_{i,t}^{last_i}$ is the latest emission date a message from flow v_i in the last node $last_i$ from path $\vec{\mathcal{P}}_i$, emitted at a date t in the first node $first_i$ in $\vec{\mathcal{P}}_i$.
- sl is the switching latency induced in each node of the network.
- $N_{i,j}^n$ is the number of messages produced by flow v_j likely to delay message from v_i in the node n. We note $N_{i,j}^n = \left(1 + \left\lfloor \frac{t + A_{i,j}}{T_j} \right\rfloor \right)^+$.

4.6 Conclusion

The Trajectory Approach, in its corrected form, represents a reliable delay computation method. It presents the advantage of being specifically designed for RT networks. It means that its implementation cost is reduced, compared to Network Calculus. This represents a main advantage, especially when defining simulation and computation algorithms.
Thus, the corrected Trajectory Approach is oriented around the flow modelling of data inside a network. We only consider, when computing a transmission delay, the different nodes encountered by a message along its path. Contrary to the holistic approach, we are not likely to consider impossible situations or irrelevant additional delays leading to high pessimism in the results.
As a result, we use the Trajectory Approach as the end-to-end transmission delay computation method in our work.

Chapter 5

Real-Time Simulation

"Un robot ne peut porter atteinte à un être humain, ni, en restant passif, permettre qu'un être humain soit exposé au danger."

"A robot may not injure a human being or, through inaction, allow a human being to come to harm."

– Isaac Asimov's First Law of Robotics [67]

Contents

5.1 Introduction

In software and hardware development, each new module has to be tested. For anyone involved in engineering or industrial design, it seems obvious to have to validate the functionalities and the reliability of new elements before deploying them. More specifically in safety critical domains, each piece of code, each new module has to be tested through different tools: automatic test generators, simulation environments, test planifications.
Implementing these tests is costful, in terms of time and ressources. The stronger the constraints to check, the higher the cost of testing it. When deploying a tool among a production environment, operating all tests directly in this environment can induce constraints of ressources and availability. Especially in the context of iterative tries, well known in software development, repeating the same test on an infrastructure a wide number of times can require to monopolize this infrastructure for long periods of time.
For example, requiring a real A380 cockpit for each AFDX network performance test can make the result of each error costful. Particularly for tests occuring during the beginning of development phases, the results of failing tests can happen to be unpredictible and implies a high fault-tolerance rate in simulation environments. What if the test fails during development phase and makes the whole test environment corrupted and necessary to destroy?
In order to answer to this problematic of availability and cost of test environment implementation, we need to define specific test environments, abstracting the highest possible number of physical infrastructures. This will allow us, for a certain number of tests, to reproduce them in an environment which is easy to setup and costless to reboot in case of failures. We need to integrate all real constraints in this test environment in order to be as close as possible to a real infrastructure. These test architectures rules out some costs and risks (human safety, mass production, ...) allowing us to increase the tolerance to faults during design and development phases of a system.
In conclusion, it appears to be necessary to conceive RT software simulators. We need to define software tools to model and represent real cases at the closest, in order to implement and test the major part of the models in virtualized environments. Once the modules have been proved fully reliable on software platforms, we can switch and start testing them on real infrastructures. This way, real infrastructures availability increased as they are needed only at the last moment of the design phase (when all tests run on virtual test environments have been validated).
In this chapter, we propose to detail the actual environment of RT and network simulation tools currently used. We want to focus on these specific tools in order to make a complete description of all constraints and needs which had to be taken into account during the simulation phase of a RT architecture. In order to be compliant to our functional requirements, we want to focus on both RT and network approaches integrated in these tools.

5.2 Requirements

We want to get a network simulation tool in order to test and validate MC models integration inside RT networks. To do this, we want to dispose of tool fulfilling specific functional and architectural requirements. Our approach in this chapter is to focus on already existing RT simulation tools and to check

whether or not they satisfy our needs.

5.2.1 Functional requirements

Functionally speaking, the tool we want to use must fullfill the following requirements :

- Schedulability analysis : We need to focus on the schedulability of simulation scenarios. We want to check if a system is able to satisfy timing constraints and the end-to-end transmission delay of a message in various cases. That also implies the system to integrate a solution to modify scheduling policies of flows.
- Network modelling : The tool main purpose should be to represent and simulate a network at runtime. That includes to propose solutions to create a graph of nodes and manage the bandwidth of the structure.
- Graphical interface and genericity : Usually, simulation tools are designed to be generic and adaptable to various kind of situations. This permits a generic approach of network simulation. But this can also represent a problem in terms of usability. In the first case, the tools are dedicated to specific uses, which makes them irrelevant when out of their context of use. In the other case, several tools are trying to adopt an approach covering a wide functional perimeter. But, usually, this approach makes the tools very complicated to use, as the user has to specify every single hypothesis of use. This problematic is common in computer science: knowing how to make balance between the size of the functional perimeter and the complexity of Graphical User Interface (GUI). We focus pedagogical purposes and so we require an intuitive GUI.
- Data modelling : To be compatible with external tools and to allow users to design simulation scenarios corresponding to their own needs, we want the tool to rely on an easy to read and modify data modelling layer. It means that the tool must integrate a solution to model simulation scenarios in an understable format and that this modelling files have to be editable.

5.2.2 Architectural requirements

We group here various guidelines we want to integrate in the software.

- Easy to use : The tool is designed to verify and check RT simulation scenarios. Its use has to be the simplest as possible, for the user to spend the less time in software management and more in configuration of its simulation scenario.
- Free : The tool we want to use is meant to be used for reasearch and pedagogical purposes. Moreover, our approach is not to build a commercial tool but to provide an open developement framework for RT network simulation to be used by many different developer communities. As a result, the tool has to be free.
- Open-Source : To provide scalability and to propose to each developer to adapt the solution to its own needs, we want an open-source tool.

- Portable : We want the tool to be implemented and used on various operating systems. Portability and easiness of installation are a cornerstone of its design.

5.3 Simulation tools

5.3.1 Functional structure

RT simulation tools are oriented to various functional contexts: performances tests, dimensioning, pedagogic approach, timing analysis, etc... In this work, we make a presentation of the main existing simulators. We want to detail their structure in order to clarify the structural and architecture choices which are common to all tools. We can regroup the different contexts of use of RT simulation tools according to the results it provides.

- The first kind of tools are designed for performance tests and benchmarking: focusing on delay analysis. This is used to establish QoS standards and to verify if timing constraints can be satisfied by a system or not. These tools are used in contexts like dimensioning infrastructures, load tests, etc...

- The second purpose is certification: the purpose is to validate the respect of constraints inside a system and provide worst case evaluations. Like performance objectives which are designed to evaluate the potential of a system, certification and constraints integration are more oriented for validating and dimensioning an infrastructure.

- A third option concerns model-checking tools, like PRISM [68] or Kronos [69]. Their role is to verify the respect of safety and reliability constraints inside a virtually modeled system. Model-checking tools were detailed in several different works like [70].

In this work, we operate a clear split between processor-oriented and network-oriented tools. The processor-oriented tools are designed for architecture simulation, containing all the tools designed to simulate and test the management of different task models in an embedded processor architecture. Depending on the tool, they can manage different hypothese of simulation (mono/multicore, shared data management). These tools are detailed in section 5.3.3.
The network simulation tools (see section 5.3.4) are designed for simulating different layers of the Open Systems Interconnection (OSI) model. In our work, we focus on an implementation of our model in Ethernet so we need an OSI layers support and integration. But we can suppose a more generical approach not relying on this hypothesis. Simulation tools can integrate dimensioning, performances and guarantees purposes. These tools are not necessarily designed to integrate RT constraints.

5.3.2 Simulation models

All RT simulation tools rely on the same fundamental model in order to model their architecture and to emulate its behavior during runtime, and this model can be detailed as composed of three different parts:

- The architecture: This contains all general parameters which are common to the different modules of the system. This allows us to define a global scheduling policy and configuration, shared memory spaces and data, etc... The architecture part of RT simulation tools modelling is responsible for defining and containing all informations which represents the meta-data of the system. This is also where we define the different external constraints to apply to the simulation context.

- The platform: This contains and describes the set of cores, processors and clusters which are going to be used inside the simulation environement (in processor context). On the opposite, this contains all the topology (nodes and links) description to model a network. The platform is the virtualizated representation of the physical devices layer that we have to simulate. The platform is responsible for the abstraction of the different hardware layers of the target system (virtual machines, drivers simulators). In this part, we can define more specifically the configuration and scheduling policies dedicated to specific subsystems or nodes and organize clusters of processors and groups of nodes.

- The data: This is the description of all tasks (grouped by tasksets) to be executed and scheduled by the system. In case of network simulation, tasks are replaced by flows. The data represents the processes to run on the system. This contains all parameters of the task and flow model (deadlines, activation models, WCET, periods, ...). The data modelling are part of the input informations of the system: it can be generated before simulation (static) or dynamically generated at runtime.

There exist two main simulation models in RT simulation: event-based and time-based. Choosing a simulation model when defining a new tool is the first important choice to make in terms of software architecture. It will impact all the representation of data and algorithmic structure of the simulation core we have to design. We propose here to detail each one of these models.

Time-based model

The time-based model, presented in [71], consists in representing a system simulated behavior according to a step-by-step evolution of the time. The implementation of the model is based on a central clock algorithm which manages the time evolution, and each new loop of the algorithm represents a new date in the system. At each new time instant, we check all the different modules of the platform (cores, network nodes) to simulate their behavior during one single time step.
The step-by-step evolution of time requires to define a time granularity for the simulation, which will represent the duration of one time step of the algorithm. This granularity can be statically-defined by the developer or can be defined at simulation configuration (pre-runtime) by the final user. As a matter, it means that time-based simulation supposes a discretization of the time model used in the simulation. We consider a fundamental time granularity which we set in the program clock (1μs step, for example), and we suppose that no event can happen between two different time sequences. The structure of a time-based simulator core is detailed in figure 5.1.
The figure 5.1 shows that the clock provides time-granularity. This granularity can be completely simulated (in RT simulation tools) or based on a real clock connected as an input point to the system. This process is frequently used in monitoring tools for industrial contexts, or in RT supervisors like PikeOS [72].

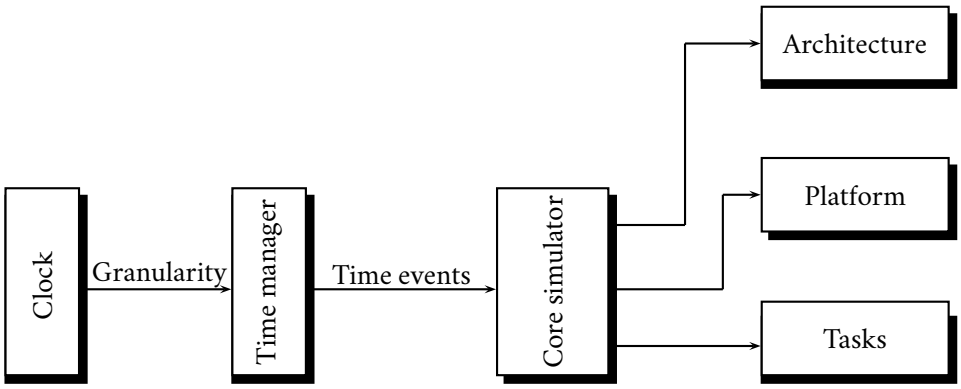


Figure 5.1: Time-based simulation

As shown, the optimism in the Trajectory Approach comes from a mis-evaluation of the serialization term bound proposed in [65]. According to the classical approach, all messages likely to delay m are transmitted in node k after M_i^k. However, due to the serialization effect, some messages are transmitted before M_i^k and their impact on the end-to-end transmission delay of m should be substracted from the serialization delay. As a result, there is an overlap between the two intervals.
The time-based modelling process presents the advantage to be closer to a classic algorithmic representation. It is easier to develop and to implement inside a RT software simulator, because its modelling is closer to classical modellings based on algorithmic loops. On the contrary, the hypothesis of discretization of the time model, and the fact that it is possible to have time instants during which nothing happens, implies a lack of performance inside the tool.

Event-based model

In order to solve the lack of performances of time-based modelling and to propose a more dynamic model, many RT simulation tools are based on a second solution called the event-based model. This model is based on the trigger-event pattern frequently used in programmation [73].
The trigger-event pattern links an external event (message incoming, new task to execute, deadline miss, execution finished, ...) to one or several dedicated function. Each module (task manager, scheduler) of the system is attached to an event handler, which triggers specific functions (control deadlines, warn the user) depending on the event triggered by the attached module. Each time an event is detected by an handler, the role of the handler is to make a link with a dedicated function in order to call it once the event trigger is detected.
Usually, event-based RT simulation tools are designed with a global event manager, connected to all event handlers. When a handler detects an event, it sends the information to the event manager, which organizes all potential events in the system. At each new incoming event, the event detector is responsible for connecting the source handler of this event to eventual destination functions. This global event detector has the role of filtering the events, and organizing them. The structure of an event-based simulation toolchain is detailed in figure 5.2.
Event-based simulation is based on a clear listing of the different events which is called the event table. This table includes and describes all events detected by the different handlers. This table can be internally

computed by the system, given the simulation context description, or it can provided by external sources through an intermediate formalization layer. For example, a sensor network connected as input to the simulator can provide regular external events, represented in a XML file. This representation of events creates an abstraction layer between the system (providing events) and the simulator (triggering corresponding functions). The figure 5.2 shows the connection between the system (Architecture, Platform, Tasks) and the event manager.

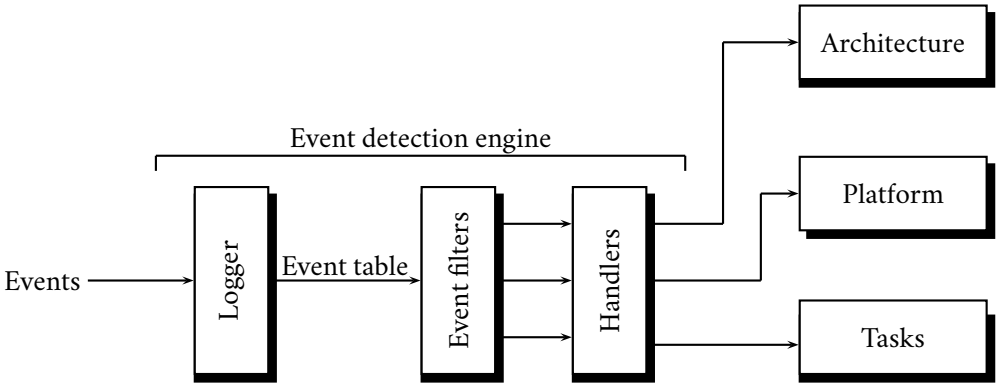


Figure 5.2: Event-based simulation

The system is connected through handlers to the event manager, which parses the event table conformly to pre-established rules. Once this parsing has been done, each event is filtered and then triggers one or different handlers. The event-based can appear to be dynamic and can propose to base a simulation on discrete or continuous time modelling. This is the simulation model of Cheddar (see section 5.3.3). This model can appear to be more convenient to simulation contexts oriented for making interfaces between real physical systems and virtualized contexts.

5.3.3 Multicore simulators

In order to understand the different RT simulation tools that are currently used, we present here major RT simulation tools. Multicore simulators do not respect the basic fundamental requirement of network modelling we need. Nevertheless, as they are a common type of simulation tool in RT domain, we want to present some of them to present some architectural approaches we want to adopt in our simulation tool.

Cheddar

Cheddar [74] is a framework designed to offer scheduling analysis and design model improvement in RT systems. Its guidelines are oriented around modularity and open-source development, making the tool easy to use and integrate inside different simulation contexts.
Cheddar has been mainly designed for two different uses: first providing timing simulation of various tasksets execution on pre-configured platforms. Each task τ_i inside Cheddar is represented as a simplified 3-tuple (WCET C_i, period T_i, deadline D_i). Based on a XML formalization of the system (taskset

and processors), Cheddar can provide a worst case execution model simulation of the taskset inside the presented architecture of processors.
On the second point, Cheddar has been designed to evaluate the schedulability of a system: compute if a taskset is schedulable or not given a set of constraints.

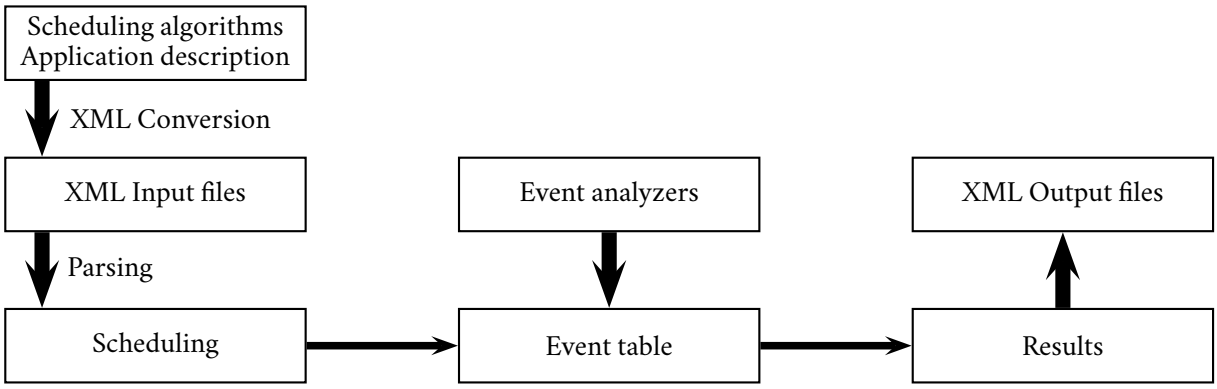


Figure 5.3: Cheddar simulator internal structure

The internal structure of cheddar (see 5.3) shows that the core is splitted in two different submodules. First, the scheduler itself, which consumes an XML representation of the system (cores and tasksets) to model. Its role is to produce the event table to give to the simulator. Starting from informations given by the system designer (the final user), the core builds a scheduling table represented by the event table. This table is encoded into XML format and then integrated into Cheddar's core.
The second part of Cheddar's core is the event table analyzer. This part contains a set of parsers and analyzers (based on rules pre-defined by the user). It parses the event table in order to represent the system behavior through the object-oriented model. Each of the used analyzers rule can be configured at each run, or predefined in the software to implement a default behavior. In both cases, Cheddar uses the analysers to extract XML data representing the final results.
XML formalization inside Cheddar's core allows us to define standard data modelling for inputs and outputs. XML formalization allows the developer to connect Cheddar with several other modelling tools such as Modeling and Analysis of Real-Time and Embedded systems (MARTE) [75]. These connections with external plugins have already been discussed in related works like [76]. This data formalization layer is the input point of all the different modules of Cheddar. It it splitted in different parts:

- The description of each processor in the system, with its different properties: preemptivity management, scheduling policies (global or local to each core), clock quantum (used for clock-synchronization [77]). This part belongs to the platform modelling.
- The specification of address spaces, which represent the address ressources and potentials of each core (text, stack, data and heap memory size, mainly). This corresponds to the description of each processor structure and architecture. This part also contains all the common properties (configuration purposes). This memory addressing part also integrates the properties used for shared data definition.

- The global description of each task in the system, with all the corresponding parameters to define each task in the scheduling context: deadline, activation model, WCET model and values, name and identifier, potential jitter, etc... This represents the task model to schedule and integrate inside the simulation core.

More details about data modelling in Cheddar can be found in related works [78], [79], [80]. The open-source and pedagogical approach of Cheddar makes it convenient for processor constraints. Its modular structure allows us to design dedicated modules to add new functionalities in its core, such as MC integration and distributed systems modelling. But that would require to modify a massive part of the data structure of Cheddar.

STORM

Simulation Tool for Real-time Multiprocessor Simulation (STORM) [81] is RT scheduling simulator written in Java. STORM is a standalone software with a GUI connected to a simulation core. It cannot be plugged to different other modules for information exchange. STORM is built to analyze the schedulability of RT systems. The modelling of a RT system with STORM consists in splitting a system in two different layers:

- The hardware layer, representing the set of processors, buses and other devices used for the simulation. This is a virtualization layer, representing different physical devices and architectures and reproducing their behavior programmatically. This layer allows us, as soon as we defined the process to virtualize an architecture, to simulate different target platforms in order to perform schedulability analysis on various architectures.

- The software layer, containing all the virtual modelling of tasks (generated by applications) and memory management. This layer is connected to the scheduling units (modeled cores and processors provided by the hardware layer) and acts as a set of virtualized drivers. Coming from these drivers, the software layer provides the simulation events to the simulation kernel. This layer also integrates all the event managers and time modelling processes inside the simulation core.

STORM integrates in its core a modulable interface based on Java modelling. This interface allows us to represent a generic scheduling policy in order to make a new external scheduler class possible to integrate in STORM core. As soon as the external class respects the pre-defined Java interface, every user of STORM can define and integrate its own scheduling policy in the tool.
Modelling input data in STORM is based on an XML representation of the information. STORM simulation parameters can be defined with a step-by-step process using this XML file. According to a root tag called *simulation*, the input file of STORM is splitted in four parts (see listing 5.1):

- The scheduling model: it links the java scheduler class to the simulation core. It describes the scheduling unit, and the scheduling algorithm used in the kernel. All the scheduling process is directly described and implemented in the external class.

- The CPU tag, describing each of the processors: java class (to implement their behavior), name, id, etc... Each tag is responsible for the individual configuration of each core.

- The taskset: each task is represented as a set of data to execute on the pre-defined platform. Depending on the system constraints, we can modify the task individual definition or make it globally by assigning global properties to the whole taskset.

- The shared data. Each task can have to rely on conccurent access to common data and this common data has to be defined properly.

```
<SIMULATION duration="75" precision="1.0" >
    <SCHED className="package.RM_Scheduler"></SCHED>
    <CPUS>
        <CPU ClassName="storm.Processors.CT11MPCore"
    name ="CPU A" id="1"></CPU>
        <CPU ClassName="storm.Processors.CT11MPCore"
    name ="CPU B" id="2"></CPU>
    </CPUS>
    <TASKS>
        <TASK className="storm.Tasks.TaskN" name ="
   PTASK T1" id="1" period="5"
        activationDate="0" WCET="1" priority="1"> </
   TASK>
         <TASK className="storm.Tasks.TaskN" name ="
   PTASK T2" id="2" period="10"
        activationDate="2" WCET="1" priority="1"> </
   TASK>
    </TASKS>
    <DATAS>
        <DATA A source="2" destination="1" rate="4"
   size="8"> </DATA>
    </DATAS>
</SIMULATION>
```

Listing 5.1: XML STORM Configuration file

The listing 5.1 shows XML data modelling in STORM. As mentioned, we can clearly identify the different parts, splitted between the tags sched (scheduling model in external Java class), Computing Unit (CPU) (definition of each processor), tasks (the processes to execute, with each deadline, period, WCET parameter) and data (defining the shared data among tasks).
The configuration integrates either a global configuration applicable to all CPUs (by applying, for example, a global scheduling policy) or to individually precise the parameters of specific CPUs.

MAST

Modeling and Analyzing Suite for Real-Time Applications (MAST) [82] is a bunch of RT simulation tools, dedicated for both processor or network RT simulation. It is composed of a simulation kernel linked with different external tools. The global structure of MAST (see figure 5.4) shows that MAST is, by itself, an interconnection of several subtools communicating either internally or through an XML modelling layer.
MAST integrates its own XML additional layer for data modelling. Similar to what we can found in tools like Framework fOr Real-Time Analysis and Simulation (Fortas) or Cheddar. The XML data description of MAST is not composed as a description of the system like for STORM, but as a listing of the different constraints to integrate in the simulation. A detailed description of this specification can be found in [82].
The internal architecture of MAST is detailed as follows:

- Data management: Dedicated graphical editor. This layer contains all the needed modules to operate a standalone usable version of MAST, able to run by itself and provide results to the user

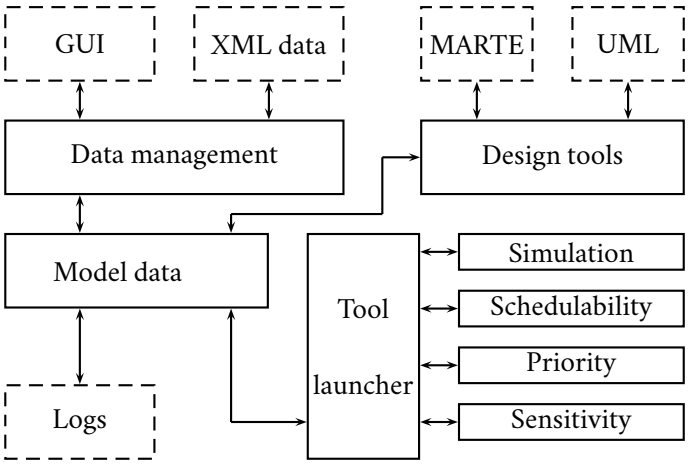


Figure 5.4: MAST Suite structure

The software architecture of MAST is detailed in figure 5.4. The core of the software is splitted in four different modules organized according to their role in the tool: Data management, Model data, Design tools and Tool launcher. We observe that the simulation core of MAST, contained in the tool launcher, is isolated from other modules. MAST also integrates, additionnaly to the GUI, a detailed log generator. It allows the user to manually detail the simulation process when the GUI is not accurate enough.
MAST is built with Ada, which integrates an object-oriented module representing the fundamental of all the software structure.

through a GUI. This part also contains the conversion module from XML and pre-formatted data into object-oriented structure.

- Design tools: This parts converts all external UML data into a reliable software model. This module regroups all the elements to represent RT constraints and modelling of a system. We can build a bridge between MARTE modelling and MAST through this module, interfacing MAST to all MARTE-compliant tools. Mainly, the Data management and Design tools are required to convert external represented data into an exploitable format for the simulation core.

- Model data: Data representation and results description. This layer operates as an intermediate between the kernel and all the external interface tools (GUI, parsers). Isolating the kernel behind a modelling layer allows the architecture to make it fully independant. This is useful to maintain this functional independance, especially when defining separated development teams about to work on the tools. The model data is responsible for converting the described system into a RT schedulable element set.

- Tool Launcher: Analyze and simulation kernel. It contains all the elements needed to operate a scheduling simulation according to the data model represented. This part contains all the RT simulation processes and schedulability analysis tools.

MAST is based on the trigger-event development pattern, which makes the kernel based on an event-oriented modelling. This model requires an event handling mechanism, in order to detect the different events inside a system and to trigger appropriate functions.
There are several types of event handlers in MAST. We have the activity, which acts as a link between a unique event and a unique output function. Activities of executing a dedicated process (linked to a piece of code) according to the detection of a specific event. MAST also integrates the multicasts, which links an event with a bunch of activities.
We can mention the multicast events and all the handlers which are responsible for manipulating and analyzing events and, on the other part, the Activities, which are responsible for execution a process (a piece of code) according to the detection of a specific event.

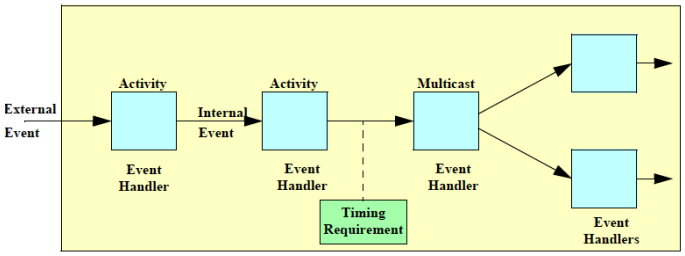


Figure 5.5: MAST event-driven model

Each event handler (Activity, Multicast) can be modeled as an analyzing box, with one input event and, potentially, one output event. Of course, the succession of events produced by successive handlers will proceed to execute different transactions representing the simulation process of MAST (see figure 5.5).

As a conclusion, we can mention that the event-driven model integrated in MAST and the high modularity inside the core allows us to model a wider set of RT and embedded systems. Mainly, MAST is a processor-oriented simulation tool, but its internal modelling allows us to do network modelling by supporting distributed systems management. Thus, the object-oriented modelling makes the development process compliant to high-level requirements, such as specific design patterns and complex functional modelling.

Other tools

Cheddar, STORM and MAST allow us to present the major design choices made when building a RT simulation. In terms of software architecture, simulation modeling, data formalization, these tools regroup a synthesis of all the potential design choices made in RT simulation. But, of course, there exist plenty of additional tools. Those who are detailed below are dedicated to more specific contexts of use: specific architecture simulation, specific scheduling policies integration, etc...
Depending on its needs and on the context of use, each laboratory, each industry designed its own tool, paying a variable attention to the existing ones. Some of these tools are public and others are totally kept as private inside institutions (either built for commercial purposes, or only internally-used). Such works like [83] tried to make an exhaustive list of existing RT processor-oriented simulation tools.
We list here complementary simulation tools which need to be mentioned, as for their importance in RT simulation tool domain or for the originality they offer in their functional approach. All these tool are designed for schedulability analysis.

- SimSO [84] is an Open-Source RT simulator for multiprocessor context. Unlike Cheddar, SimSo is based on a Python architecture, modelling input and output data flows as Python instances and classes. It is designed to be an easy-to-use simulator, with a fundamental representation of a RT system which can be entirely configured.

  The main representation of a system in SimSO is based on presenting a RT multicore context as a set of tasks to be run on a specific platform. Tasks are represented by their unique identifier, WCET, period and activation date, according to the model presented in chapter 2.

  In SimSo, each processor is basically represented as a couple (id, name). Considering that SimSO is oriented for overhead management, processors can be individually configured to integrate switching time and scheduling decisions. SimSO allows the user to upload its own scheduling policies, written as Python classes, dynamically in the simulation engine.

Basically, SimSO is based on a discret-event simulation framework made in Python and called Simpy [85]. Simpy is a framework which embeds all the basic modelling needed in the RT paradigm. Simpy allows the user to implement the task and job modelling required in SimSo to operate a scheduling simulation.

- Fortas [86] is a Java-oriented framework designed to analyze and compare the schedulability of RT systems, based on analytical test results. The internal architecture of Fortas makes it able to consider partitioned and semi-partitioned approach in multiprocessor simulation context.

  Fortas is a an Open-Source test-oriented tool, designed for reasearch and educational purposes first. The duality between Java and XML in its core integrates high interoperability inside the software.

- SchedMCore [87] is also an Open-Source set of tools designed for schedulability analysis and performances evaluation inside RT systems. SchedMCore is based on an internal formalization model, generated from Uppaal [87] network modellings. It proposes to configure each scenario through a light-syntax file format. This increases the usability of the tool and limits its ressources cost (the input files are light weight).

  The interoperability between SchedMCore and Uppaal allowed the developers to conceive an easy to use tool. SchedMCore has been designed for simple access to simulation and schedulability results. The results provided by the core are easy to read and parse in order to be integrated as input to external tools. This integrates interoperability and, also, the results are easy to compare when targetting benchmarking purposes.

Comparative approach

In order to compare different RT simulation tools in terms of architecture and data modelling, we based our work on different extracts from the litterature to build the following comparative table (see table 5.6). This table centralizes all the RT simulation tools which are still maintained and which could be used as a reference in terms of RT simulation and data modeling.
Based on these results, these are the different elements we want to have in our simulation process :

- XML data modelling : Presented in STORM and Cheddar, it appears that XML is a proper format to model data, even if it is not the lightest one. It presents the advantage on being easily readable and it offers modularity. This will allow developers to define specific formalization standards depending on their own needs.

- Inter-operability : Mostly found in MAST, the potential to connect a simulation tool to various external tools (MARTE, user interface) is a fundamental requirement.

- Java development : Most of the tools we presented are based on Object-oriented languages (Python, Java, Ada, C++). To fullfill portability and to open to a high number of developer communities, Java appears to be the most relevant choice.

Name	Data Modelling	Language	Analysis	Simulation	Open-Source	References
Cheddar	XML/AADL	Ada/C++	Yes	Yes	Yes	[74]
Fortas	XML	Java	Yes	Yes	Yes	[86]
Yartiss	XML	Java	No	Yes	Yes	[88]
MAST	XML/XSD	Ada	Yes	Yes	Yes	[82]
STORM	XML	Java	No	Yes	No	[81]
CPAL	CPAL	CPAL	No	Yes	No	[89]
SchedMCore	Internal	C	Yes	Yes	Yes	[87]
pyCPA	Internal	Python	Yes	No	Yes	[90]
SimSO	Internal	Python	No	Yes	Yes	[84]
SymTA/S	Internal	Commercial	Yes	Yes	No	[91]
TIMES	Internal	Java	Yes	Yes	No	[92]

Figure 5.6: RT Simulation and Analysis tools

5.3.4 Network simulators

We detail here major network simulators, based on the previous work established in [93]. RT simulation and network simulation are not necessarily related, especially in simulation tools. The tools we present here do not all integrate RT network architectures nor rely on RT protocols like AVB, AFDX or CAN. In network simulation context, the main tools tend to focus around performance and infrastructure dimensioning, more than scheduling analysis.
Additionnally, the presented tools could be more industry oriented and less designed for pedagogical or research purposes. This can involve a lack of configuration solutions, logs reachability and modularity. Our work is to focus on these different tools in order to extract their simulation and modelling solutions, in order to combine them with the RT approach we need to adopt in our work.

NS

Network Simulator (NS) [94] is an open-source network simulator designed to interface virtualized topologies with physical infrastructures, in order to provide network virtualization and to simulate network dimensioning problems. It is based on open modelling standards, making it generic and easy to plug with different network tools: traffic generator, network listener, etc... The virtualization model of NS and the different communication interfaces it provides integrates various network devices inside the simulation core, making it compliant even to specific or emerging technologies.
NS is based on an object-oriented structure (C++/Python), which implies a hierarchical class organization. Its internal class structure is based on a duality between C++ and OTcL objects [95]. OTcL is a library designed especially for NS. It is an extension of the TcL library [96] built for scripting integration into real-time and embedded systems simulation. NS is based on the combination of these two languages: C++ is used for runtime management and efficiency, as OTcL is used for simulation management and configuration. This duality can appear to add more complexity in NS at the first time, but it assures a clear separation between network protocol implementation (written in C++) and simulation context (OTcL). This makes the tool easier to main and to improve: dedicated developers can be attached to each

module, without requiring to master both C++ and OTcL technologies.
The recent versions of NS (NS-3) integrates communication interfaces with python scripting modules, allowing us to model simulation configuration directly with Python classes, which is more compliant to open-source standards. NS can be used in different contexts: industrial, commercial, research... As a conclusion, its users are not necessarily familiar with configuration files edition. In order to answer to this problem, NS integrates a light dedicated syntax to define simulation scenarios.
The main purpose in NS design is to integrate performance tests and traffic management inside a network infrastructure (see figure 5.7). Like it was showed in [97], precision and performance improvements have been integrated into NS to emulate dynamically a network behavior, converting real infrastructure inside the discrete event-oriented kernel of the software.

NS allows to define a node and its behavior on different layers of the OSI model, mainly dedicated to define standards between physical and session layers. Each node can be configured at a different level of details, allowing the user to create subnetworks of various genericity. As shown in figure 5.7, each software-created node can be interfaced with a virtual driver and then plugged as an input or output in a port of a real network commuter.
This hardware simulation integrated in NS also makes a conversion between the discrete time modelling in NS kernel and RT communication through nodes inside a real physical topology.

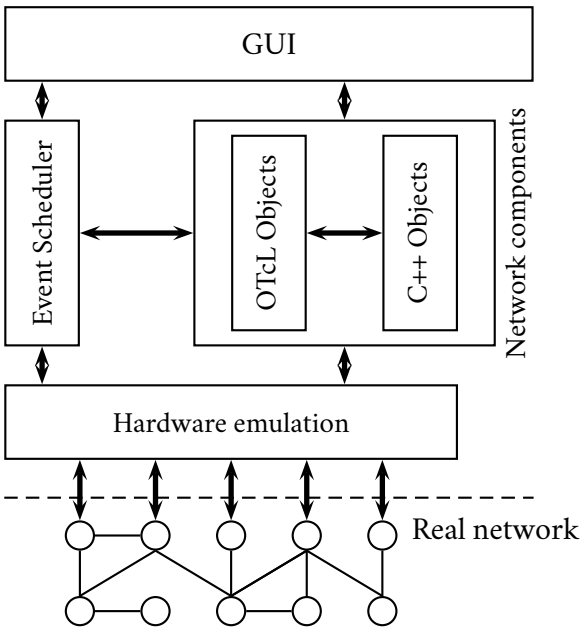


Figure 5.7: NS emulation model

The figure 5.7 shows the global architecture of NS (detailed in [94]). The network modelling in splitted between oTCL and C++ objects. All objects behavior are scheduled and time-organized according to the event scheduler, whose role is to manage events handling and triggering during runtime.
NS is an open-source software which represents switches and nodes corresponding to all kind of network protocols in the different layers of the ISO model [97]. It can simulate different network physical mediums (wire, wifi, cellular), and implement specific protocols on each. Classical standards of IP networks are already included in the standalone version, and the modular structure of the software allows to integrate different other protocols and mediums. NS represents and simulates specific network architectures designed for RT domain, such as CAN, AFDX or AVB by integrating dedicated external modules. It means that NS is not a real-time simulator by itself, but it can integrate RT constraints. For example, we can simulate IEEE-1588 compliant switches with specific IEEE-1588 modules integrated in the hardware simulation layer.

Why not NS ?

Thus, NS is not dedicated to schedulability analysis. Its modelling of networks implies defining each node at its physical level and each flow at frame level. This can represent an unnecessary loss of time. Thus, the hardware layer relies on the implementation of virtual drivers to model specific physical devices, which implies disposing of the complete specifications of each device we want to implement.

OMNeT++

OMNeT++ [98], [99] is an open-source tool. It is a C++ object-oriented modelling framework used for network simulation. It is based on a component-approach structure: each part of the tool is developed separately. The architecture of these components is detailed in figure 5.8. Its structure is comparable to Cheddar: it contains its own GUI which induces an ergonomical use but is modeled as a set of independent tools (simulation framework) and not as a standalone simulation software.

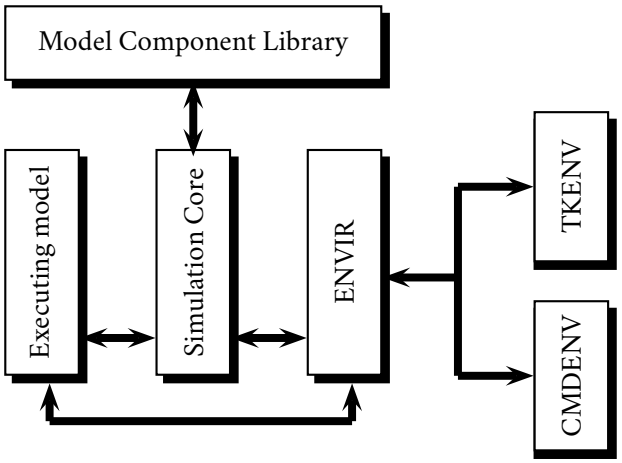


Figure 5.8: OMNeT++ architecture

The figure 5.8 details the internal architecture of OMNeT++, detailed in [100]. This architecture is based on a dual communication interface with the user. OMNeT++ integrates a GUI based on TK library [101] and also a command-line interface allowing to directly create TCL objects.

The architecture of OMNeT++ tool is composed of the following main components:

- ENVIR: This module is responsible for analyzing and parsing all the input informations destinated to the core. Its role is to build all the network topology and integrate the modelling constraints.
- CMDENV/TKENV: These modules are the two potential user interfaces. CMDENV integrates a command-line TCL configuration, as TKENV provides a GUI for a more ergonomical approach. The GUI is a totally independant module which can be splitted from the OMNeT++ core without impact on other modules.
- Model Component Library: Additional libraries with component-oriented structure like INET [102] and Castalia [103].
- Executing model: This contains all the constraints and configuration related to runtime management and simulation implementation.
- SIM: this represents the simulation core, emulating the network behavior during runtime.

OMNeT++ is built with a component-by-component approach to improve its modularity. In order to simplify the network configuration in OMNeT++, topology modelling is based on a dedicated language, called Network DEscription (NED). NED is a description language which was presented in [104], [105]. The purpose of NED is to improve the accuracy and configuration potential of OMNeT++ without impacting the ergonomy and usability of the tool. In order to do this, NED relies on a component description interface, allowing to describe each node through a dedicated NED file, and to connect it to other nodes like LEGO bricks, by defining each element connected as input or output to others. Through a simple and verbose syntax (see listing 5.2), each component of the topology can be precisely and easily configured.
The purpose of the NED language is to define, describe and assemble all network components through a set of predefined-properties. Each node, link, data frame of the network can be defined through NED language. The NED language integrates a structure extracted from object-oriented development. We can cite, among all, inheritance, prototyping, hierarchical structure and interfaces. NED is a fundamental component from OMNeT++ core as it allows the tool to connect all the parts together and to model the simulations to run.

```
network Network
{
    types:
        channel C extends ned.
    DatarateChannel {
            datarate = 10Mbps;
        }
    submodules:
        node1: Node;
        node2: Node;
        ...
    connections:
        node1.port++ <--> C <--> node2.
    port++;
        ...
}
```

Listing 5.2: NED syntax example

NED is similar to Architecture Analysis and Design Language (AADL) in terms of design. It is an object-oriented description language based on a hierarchical structure. In integrates such concepts as inheritance, interfaces, packages organization and metadata management. An example of a basic two nodes network modelling is given in the example 5.2.
The example of listing 5.2 shows a simple network composed on two nodes (Node1, Node2) linked through a 100Mb/s channel. Each node behavior is described by another NED file of type Node.

OMNeT++ integrates user-oriented solutions to manage simulations, particularly through the GUI and the intuitiveness of the NED syntax. Either the context is simple and it can designed through the GUI, or the user can have more specific needs and edit manually the NED configuration files. In that case, the presence of the NED language allows the designer to create an communication interface to the simulation core which opens a wider potential of simulation configuration.

Why not OMNeT++ ?

OMNeT++ is designed as a framework for network simulation tool. It can be seen as an Integrated Development Environment (IDE) for network simulation, with a very high modularity. Nevetheless, the tool is not designed to integrate RT constraints such as the potential to modify the scheduling policy of network devices.

OPNET

OPNET [106] is a commercial network simulator, designed for industrial and commercial purposes. It can be assimilated as the commercial alternative of OMNeT++ [98]. OPNET has been designed for the management of a wide range of potential networks from personal LAN to global satellite networks. Similarly to NS, it allows the user to create interactions between networks based on different mediums. For example, we can focus on the ressources costs implied by the interaction between a national cellular network dedicated to GPRS/GSM implementation and a LAN dedicated for configuration inside a specific phone company.
The main purpose of OPNET is to provide an easy-to-use commercial tool to propose implementation and dimensioning solutions to industrial problems. The design of OPNET does not necessarily orient it to IP or even wired networks. On the contrary, it is not designed to cover all the different layers of the ISO model [106]. The network simulation model of OPNET targets allows the user to act at each level of the OSI model, making him able to modify essential physical parameters of the topology implementation.

OPNET design has been designed to privilege the user experience, in order to open the usability of the tool to non-specialists. Additonnally, this design choice allows to focus directly on simulation results without implying a potential traduction phase from the user. All provided results are explotable directly through a GUI (see figure 5.9).
Unlike NS or OMNeT++ which were structured as modular frameworks, OPNET has been designed as a turnkey tool. This has been detailed in terms of precision, performances and ergonomy, but the functional perimeter is limited by the updates the developer team can provide, depending on the commercial support of the tool.

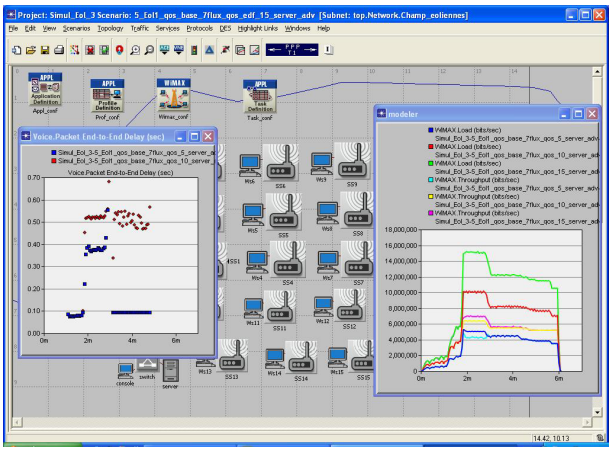


Figure 5.9: OPNET User Interface

In order to be easy to manage by the user, configuring a simulation with OPNET has to be done following a specific sequential process. This keeps the configuration environment clear and easy to structure. This process is detailed as follows:

- Define problem: We specify the system architecture and the simulation problem we have to solve. This first step consists in defining the functional perimeter of the problem.
- Build model: Starting from the previously defined informations, we model and integrate them inside OPNET through the GUI (see figure 5.9). We transpose the constraints into software functionalities configuration.
- Simulation: Once the model has been clearly defined, we launch the simulation on runtime. We can then observe the evolution of the network along time. OPNET provides runtime management functionalities (pause, resume), mainly for analysis and debug purposes.

- Analyse results: Once the simulation is stopped, we exploit the results to compare them to the defined model. These results implies defining a clear way to model and represent the results through a specific logger module. OPNET provides different additional modules to complete the results analysis (grapher, stats, logs).
- Make decisions: Once the results have been exploited and analyzed, we can decide to implement the described topology, or to adapt the simulation context in order to integrate additional constraints or modify the given configuration.

This sequential execution model is common to many RT software tools and, more generally, to a wide part of commercial simulation softwares. This simulation process comes from project management methods like Plan Do Check Act (PDCA). It is well adapted when working on dimensioning on performances purposes (where we want to adjust input paramters until the right configuration can be found), but it is not designed for certification. It would require more details on decisions to make and integrate solutions to run successive similar simulations for benchmarking and validation purposes.

Why not OPNET ?

OPNET can be considered as a simulator dedicated to help network designers to take specific decisions before and during physical implementations. Even if this is detailed, this is far from the schedulability analysis and RT design we want to focus on. Thus, the virtualization of the concepts does not allow to edit specific frames format, forbidding personal protocols. The tool does not fullfill data modelling and open-source requirements.

AFDX Tool

Cheddar, STORM or NS are adopting the approach of increasing genericity at the expense of ergonomy. The tool we present below proposes a web-oriented ergonomical tool for AFDX network monitoring.
This AFDX monitoring tool was presented in [107]. We detail here some major points of its design. The tool is based on a web-oriented architecture. This assures a portability of the software on any potential server and it allows the user to integrate technologies dedicated to ergonomy, such as the CSS language. The web-oriented architecture allows the AFDX monitoring tool to make independant the network simulation core and the GUI part of the software, run on the web clients.
The architecture of the tool is detailed in figure 5.10. The server part is connected the the AFDX through a free port of a given switch in the AFDX topology (see chapter 3.2 for details about AFDX topology). This connection between the web server and the AFDX allows the tool to get online data about the network's behavior and state.
The second purpose of the web-oriented architecture is to allow users to simultaneously access to the same data at the same time. The AFDX monitoring tool can be used and configured by several users at the same time. All computers connected to the same LAN as the webserver can access to the monitoring data through a GUI called the visualizer. This visualizer is the client-oriented part of this web-oriented monitoring tool.
The role of the visualizer is to parse the file sent by the server (formatted in JSON [108]), got from a set of asynchronous requests sent with AJAX. The server provides a dynamic live JSON Application Pro-

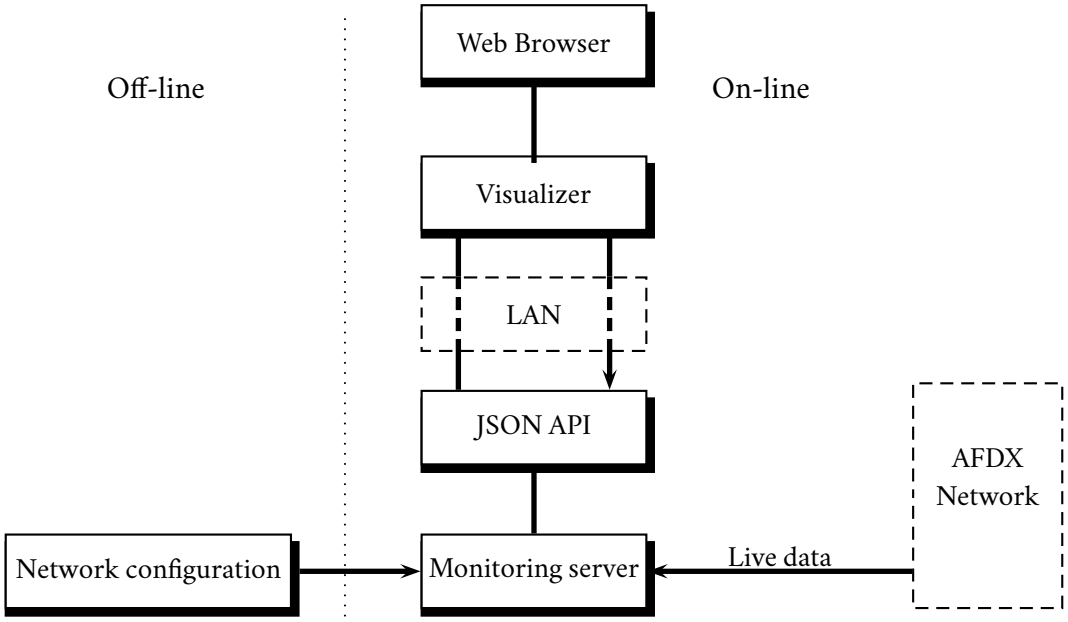


Figure 5.10: AFDX Monitoring tool architecture

grammable Interface (API) for the different clients, which act as JSON parsers to organize and displays data on client browsers. This API allows the tool to propose a mix between the efficiency and potential performances of an object-oriented language, and at the same time to offer the ergonomy proposed by web-oriented design. As long as the server's access is reachable, each client can connect and communicate with the JSON API.
The architecture of the tool splits the information in AFDX network in two classes: off-line (configuration, network description) description and on-line live information (messages transmission, topology information).

- The off-line informations are uploaded in the tool by the user through the visualizer. It contains all the network configuration and description of the different constraints applied on the network traffic. It is the static modelling of the network model.

- On the opposite, the on-line part contains a detailed description (generated at runtime directly by the AFDX network) of all data managed by the network at a given instant, or according to specific events. These data are dynamically generated during the simulation.

All these informations (on-line and off-line) are stored in the dynamically generated JSON files. This files are an interface betweeen AJAX functions (for the web part) and Java functions (for the core) [109]. The JSON files are generated with a algorithmic kernel and these files are used as a link between final application and data sets (database or computation results). This structure presents the advatange of abstracting the kernel from all the external module and applications: no matter the role and design of the external modules, the kernel is not impacted by their architecture. The developer just has to design the format of exchange files.
The purpose of the tool is to monitor and manage an AFDX network. It means that the user is authorized to upload its own configuration data to the server through the visualizer, data which will be sent then to

the AFDX network at runtime. These datas uploaded by the user are considered, from the server, as off-line network configuration informations. The server is then responsible for mixing on-line data (from AFDX) and off-line data (from the user), as shown in figure 5.10. This approach also allows developers to split the different teams (PHP/Java/Ergonomical design) working for the tool, in order to simultaneously simultaneously the different parts of the software.
As a conclusion, relying on client-server architecture for user management and on object-oriented structure for simulation and monitoring integrates a strong functional and architectural separation between the different services of the software: configuration, data management, traffic and network control, ergonomical display. This approach can be reused in order to be applied to other tools.

Why not AFDX Tool ?

The architectural choices made for this tool are convenient to our requirements. Nevertheless, it is strictly designed for AFDX architectures and, as a result, does not respect the generic approach we want to fullfill.

Conclusion

In network simulation, the usability and ergonomy takes a strong place inside this context. Moreover, we can observe that all the presented tools have these elements in common:

- Data modelling is standardized (XML, JSON, ...). Each representation of a data (network, message, node, ...) is defined in a specific language and respects a clear enounced structure. This differentiates each independant module by creating a sub-api dedicated to module communication.

- There is a clear separation between the runtime environment manager and the simulation modeler. It means that the modules in the software responsible for modelling are separated from the modules which are responsible for simulation. This integrates and models data independently of the simulation.

- Communication interfaces and exploitability of the results are specifically detailed. Each tool integrates a GUI, log managers and specific parsers in order to be able to translate the input and output from various set of sources: XML, internal language, graphical definition, etc...

In order to focus on MC impacts in RT networks and more specifically to analyze scheduling scenarios integrating MC, we want to use a simulation tool able to mix these different approaches. Even if the presented approaches can be reused, each presented tool presents specific lacks in terms of modelling, usability or ressources costs which makes it unusable for our specific context.
Considering this state of the art about RT and network simulation tools, we designed our own tool based on some of their architectural and functional choices, but dedicated to our need. The design and development of this tool has been a major part of our work. It has been detailed in part IV.

5.4 Task modelling

The representation of tasks and flows, respectively for processor simulation and network simulation, relies on the definition of specific parameters. For example, each task can be represented by a fundamental couple $\{C_i, T_i\}$. These are the basic properties of a task. In the following part, we propose to focus on solutions to model and create tasks (or flows) in RT simulation tools.

5.4.1 Execution time simulation

To assure reliability and guarantees, the most common model to represent a task is to model it with its WCET. This allows tools to provide simulation results showing the worst case analysis of a system. But modelling a task delay just as its WCET is not complete: all simulations contexts are not necessary oriented for worst case studies. There exist different simulation contexts where WCET value is not sufficient to characterize a task execution delay.
As a result, one additional information which is commonly added to the task model is its Best Case Excution Time (BCET). BCET and WCET of a task are the respective bounds of a task execution time in the model. It means that, during runtime, we can guarantee that there is no situation where a task execution time will be below its BCET, or beyond its WCET.
Worst case analysis provides simulation results which are reliable and useful for certification purposes. But, when targetting different purposes, results provided by worst case analysis can appear to be very pessimistic. In terms of modelling, introducing BCET implies modifying the model of a task: its execution time is no more static, but the execution time of each job is bounded. In our work, we model a task execution time according to various values, all indexed on the WCET of a task. The BCET and WCET values model an interval in which we define different values corresponding to different criticality levels. This approach has been detailed in chapter 6.
Even if this occurs to be comparable mechanics inside software simulation kernels, considering different approaches in terms of execution time represents a real functional difference. This does not correspond the same analytical approach to do on the results: this provided results which could be closer to real implementations, and more far to theoretical modellings. Also, integrating this solution implies defining models to decide what is the value of each execution time, at each job or task release time.

5.5 Random tasks generation

The simulation structure of a tool can be synthesized, basically, as composed of schedulers (CPUs) and elements to schedule (tasks, flows). Depending on the context, these elements can integrate various constraints and additional hypothesis, but this is the basics to find in any RT scheduling simulator.
It means that above all else, we need to be able to define tasks (or flows, for network context) which are supposed to be executed (or transmitted) on the defined CPUs (or nodes). When defining a simple example for simulation, these tasks can be statically defined by the user. We focused on this previously: by using dedicating files or through a GUI, the user can define the taskset of its simulation. In that case, the user will define one by one each parameter of eacha task (deadline, period, WCET). This context of use is mainly used to build an demonstration example or focus on a typical case of simulation.

Creating manually simulation scenarios and filling task parameters can quickly appear to be tedious, particularly when generating successive tasksets. Basically, the main purpose of these simulation tools is to offer reliability and performance on their results, for example with experiencing hundred thousands different tasksets (with different parameters) on the same platform, in order to analyze a more exhaustive field of results. To obtain these results, we have to generate mass of tasksets. It is obvious that we cannot anymore build them manually, for time and accuracy reasons. That is why we have to define tasksets generators which will operate as tasksets builders for our simulation context and platform. The point we want to focus here is about the methods used to generate random tasksets in RT simulation tools and how to apply these methods to flow generation for networks.

5.5.1 UUniform algorithm

Intialization

The RT simulation mentioned previously mentioned, particularly SimSo [84], Cheddar [74] and Fortas [86], based their taskset generation model on the UUnifast algorithm [110]. This is a common taskset generation algorithm based on a uniform approach. We want to present here the fundamentals of this algorithm. It is a major generation algorithm in RT simulation.
The main goal of UUniform is to generate a valid taskset of size n for a RT simulation. To be considered valid, a taskset must correspond to several different constraints:

- The size of the set has to be of n tasks, with n a static user-defined value.
- Each generated task t_i must be defined by a period T_i and a WCET C_i, and must respect the condition $C_i \leq T_i$.
- The global utilisation represented by the taskset to generate, computed as $\mathcal{L} = \sum_{i=1}^{n} \frac{C_i}{T_i}$, has to be statically defined by the user and initially fixed before the generation process. The value of $\mathcal{L}$ is called the load of the taskset.
- The period of each task must be upper-bounded. This bound has to be lower or equal to the length of the time interval during which the simulation occurs. We note this bound as T, and we have: $\forall i \in [1;n], T_i \leq T$. This applies only for sporadic and periodic flows.

The input parameters needed to generate a valid taskset are: the size n of the set, the targetted load $\mathcal{L}$ and the maximum period length T. These are the input parameters of UUniform.

Generation process

In terms of model, UUniform generates a set of n tasks, each task represented by a couple $\{C_i, T_i\}$. The purpose of the algorithm is to generate a set of n tasks, all defined by a specific period and WCET. We suppose that the simulation context we focus is based on a discretization of the simulation time interval. It means that this time interval is defined by a time granularity T_g. It implies that: $\forall i \in [1;n], \exists k \in \mathbb{N}, \exists T_i = k * T_g$.

The first step of UUniform algorithm is to generate a set of n periods $T_1, T_2, ..., T_{n-1}, T_n$. These periods are generated according to a uniform law $\mathcal{U}$. They are all included between a minimum T_{min} and a maximum T_{max}, statically defined as inputs. According to what we defined previously, we assume that $T = T_{max}$. First, in order to generate a task τ_i, we compute a value r_i, which is the fundamental expression of the period, extracted from a uniform distribution. Then, we deduce the value of period T_i with the following expression:

$$\begin{aligned} r_i &= \mathcal{U}(log(T_{min}), log(T + T_g)) \\ T_i &= \lfloor \frac{e^{r_i}}{T_g} \rfloor * T_g \end{aligned} \tag{5.1}$$

Based on the randomly uniformly-generated periods we obtained, we do not directly compute the value of the WCET of each task. We first compute the utilization value u_i of each task. This utilization is generated according to a uniform law included between 0 and 1. Each value of u_i is computed with:

$$u_i = \mathcal{U}(0, 1)$$

This value gives us the individual utilization represented by each task. In order to check the validity of the taskset, we check the value of $U = \sum_{i=1}^{n} (u_i)$. If we have the following constraint verified: $U = \mathcal{L}$, the taskset is validated. Otherwise, we discard the taskset and we regenerate a new set of utilizations.
If the generated taskset is validated, we finally have to deduce the different WCET from the following expression:

$$C_i = T_i * u_i$$

Considering this expression, we finally obtain a set of n couples $\{C_i, T_i\}$. This corresponds to the algorithm 5.11.
As a conclusion, UUniform algorithm provides an easy to implement solution to generate a taskset of size n. UUniform algorithm is based on a discarding logic when generating the tasksets. The current process adopted in UUniform is to generate a complete taskset according to the input parameters and, finally, to focus on the taskset. If it complies to the needed constraints in terms of load, we keep the taskset. If the final load is not correct, we discard the taskset before re-generating a totally new one.

5.5.2 Discarded tasksets correction with UUnifast

Uuniform generation process allows the tasksets generators to be accurate and reliable. We can assure that the generated taskset will be of the correct the load and number of tasks, asked as inputs. But, the current generation process of UUniform has a high number of discarded tasksets per generation loop.

```
Data: n_t, n, L_min, L_max
T ←
U ← 0
while U ≠ L do
    U ← 0
    for i ← 1 to i ≤ n do
        r_i ← U(log(T_min), log(T + T_g))
        T_i ← ⌊e^{r_i} / T_g⌋ * T_g
        u_i ← U(0, 1)
        C_i ← u_i * T_i
        τ_i ← {C_i, T_i}
        U ← U + u_i
        T ← T + {τ_i}
    end
end
```

Figure 5.11: UUniform taskset generation

The UUniform taskset generation algorithm is built on a task-by-task model. Each task is generated independantly and corresponds to a randomly computed period. Depending on the uniform law $\mathcal{U}$, we generate a specific utilization for each task, and then we deduce the WCET of the task.

The value of r_i has been detailed in [110]. r_i is a random value, computed according to the uniform law $\mathcal{U}$. All the generated values of r_i are included in the interval $[log(T_{min}); log(T_{max} + T_g)]]$. The value of T_{min} has to be defined before generation. It represents the lowest value of a potential period. This value can be set, defaultly, to T_g.

The expression of T_i, depending of the value of r_i is: $T_i \frac{e^{r_i}}{T_g} * T_g$. Given the interval to which r_i belongs to, this expression guarantees that T_i will verify $T_{min} \leq T_i \leq T_{max}$ for each task τ_i. The uniform law guarantees a heterogeneity in the generated periods. All the generated tasks will have a different execution time profile, with periods of variable length.

This can happen to be very costly in terms of performances and time. On a single taskset generation, this cost is hardly noticeable, as the generation time of just one taskset is low (from the point of view of the user). Each generation time can be of an average time of 10 miliseconds. A difference of a few milisecond won't majorly impact the user experience.

But in RT simulation context, the process used in simulation tools such as SimSo or Fortas implies generating wide group of tasksets, which can be composed of 100.000 tasksets or even more. As a conclusion, the discarding logic can strongly impact the performances of the generator. Each few milisecond difference can, in this context, represent a huge impact in terms of simulation performances as this difference will be reproduced for each generated taskset.

Simulating RT scheduling scenarios implies relying on a discretization of the time model. This discretization can introduce a potential error margin, making useless to reasearch a too high precision in WCET evaluations. FFor example, a gap of load of 0.001 can represent, in a concrete case, a difference of transmission time lower then $0.6\mu s$ in WCET evaluation, which is below the minimum size of a message (on a classical 100 Mb/s Ethernet network) and can be considered as negligible.

As a conclusion, one possible solution introducing lack a accuracy would be to permit an error margin on the computed utilization. This error margin ϵ will induce that the condition on U will be then $\mathcal{L} - \epsilon \leq U \leq \mathcal{L} + \epsilon$. As soon as ϵ can be strictly defined and its value stays manageable, this can reduce the tasksets generation time.

But integrating an error margin is not a reliable solution in terms of efficiency and reliability of the generation model. Thus, this error margin integration will quickly show its limits and will not be sufficient to significantly increase the taskset generator performances. In order to solve this problem, an alternative algorithm to UUniform has been introduced. This algorithm, called UUnifast, is detailed below.

Discarding rate improvement

As shown in [111], the probability for the utilization U of a taskset generated with UUniform to not exceed $\mathcal{L}$ is lower than $1/n-1!$. This is strongly impacting the performances of the generator. It means that, in average, only $1/n-1!$ among all generated tasksets are validated.
If we work with small tasksets for demonstration purposes ($n \leq 10$), this can stay manageable. But if we suppose that we want to generate tasksets of 50 tasks, the number of discarded tasksets (and so, of useless algorithm loops) will be of $30.10^6 3$. It seems obvious that this is way too high, even for an efficient generator.
In order to answer to this problem, a improved version of UUniform has been designed, which is called UUnifast. UUnifast generation relies on the following observation: as the probability to generate a valid taskset is higher, the time needed is lower. The purpose is to provide a taskset generation algorithm with a lower discarding rate.
UUnifast taskset generation is based on a sequence S intialized to $\mathcal{L}$ and where each successive term is computed with a random uniform value, depending on the task index. To do this, we operate on the computation of the value of u_i for each task τ_i, which is no more generated according to a simple $\mathcal{U}(0,1)$ uniform law.

$$\begin{aligned} S_n &= \sum_{i=1}^{n} * u_i \\ S_{i-1} &= S_i * \mathcal{U}(0; 1/(n-i)) \\ u_i &= S_i - S_{i-1} \end{aligned} \tag{5.2}$$

The more i increases, the highest is the potential value of $\mathcal{U}(0; 1/(n-i))$. But, on the contrary, the sequence S tends to decrease with the value of i. As a conclusion, the value of u_i tends to be balanced and less purely random as for UUniform.
Once we generate the utilization u_i for each task τ_i, we just have to deduce the corresponding values of C_i with the expression $C_i = U_i * T_i$ (same as UUniform). Next, we can compute the global utilization U of the generated taskset, and check the individual utilization of each task. If we have $s_n = \mathcal{L}$ then we consider the taskset as valid. In the other case, the constraints are not respected, we discard the generated taskset and we create a new one, keeping the previous discarding process.

Simulation results

The difference of discarded messages rate between UUniform and UUnifast is shown in figure 5.12. We can observe that the number of discarded tasksets drastically incrases in UUniform as we need to generate a bigger taskset. As shown in [111], the acceptance ratio of UUniform can be upper bounded by $\frac{100}{(n-1)!}$. Figure 5.12 shows a comparison between UUnifast and UUniform algorithm for generation of tasksets between 3 and 10 tasks. Each point is the result of 100 algorithm loops.
The curves of figure 5.12 show that, for small tasksets ($n \leq 6$), Uuniform algorithm has a better acceptance ratio. But, as soon as we target to generate tasksets of size $n > 7$, the acceptance ratio of UUniform tends

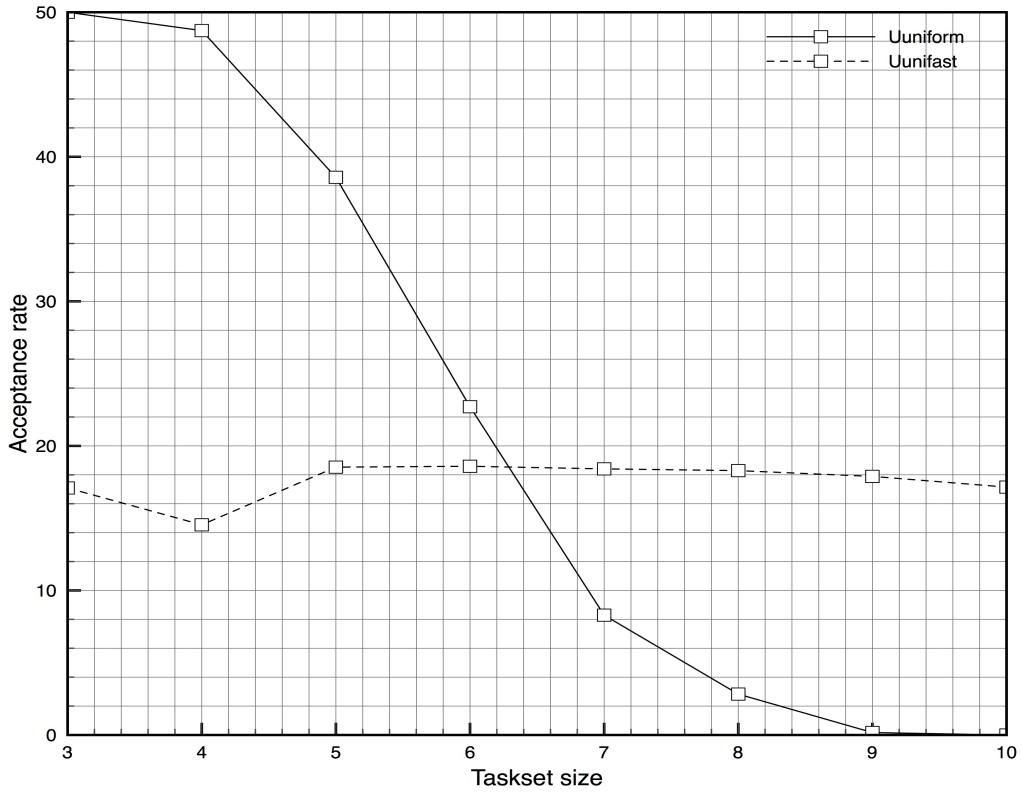


Figure 5.12: UUniform and UUnifast acceptance ratios

to decrease below 0.5 %. Thus, UUnifast tends to assure a stable acceptance ratio, around 17 %.
We also tested that, when n exceeds 15, the acceptance ratio of UUniform decreases below 10^{-8} which is far too low and represents a strong impact on execution time of the taskset generation program. That is why UUnifast strictly improves the generation process, keeping a reliable acceptance ratio even at high taskset sizes.
As a conclusion, UUnifast is a reliable taskset generation algorithm, which is less likely impacted by the targetted taskset size in terms of performances. It is well adapted to generating wide tasksets. Particularly in network context when generated flowsets can increase to 50 and beyond, its generation process has to be kept.

5.6 Conclusion

There exist RT and network simulation tools and models based on different approaches of the RT domain. Performances, analysis, dimensioning are all functional targets, based on different simulation models. But, despite this wide set of tools, MC has not been integrated in any of these tools as an highlighted functionality.
The purpose of our work would to be to integrate RT and network simulation tools architectural concepts inside a new tool, designed for schedulability analysis in MC scenarios.

Part III

Mixed criticality management protocols

Chapter 6

Mixed Criticality

"Si vous n'êtes pas capable de l'expliquer à un enfant de six ans, c'est que vous ne le comprenez pas."

"If you can't explain it to a six year old, you don't really understand it."

– Richard Feyhnman

Contents

6.1 Criticality in Real-Time scheduling

The purpose of priority assignement is to attach a level of importance to each flow. Priority assignment and priority-based scheduling represent the fundamentals of RT scheduling and RT networks scheduling. This topic has already been discussed in various works like [16], [112].
In RT domain, assigning a priority to each flow consists in defining a value $p_i \in R^+$ for each flow v_i. We assign a different priority to each flow according to policies and scheduling constraints, and we schedule the flowset transmission order in a node according to this priority assignment. This is the same process as in processor context. There exist in RT networks many different scheduling policies, many of them inherited from RT processor context: FIFO, Fixed-Priority [112], Rate-Monotonic [4], Earliest Deadline First [113], etc... each policy belonging to different needs and design constraints.
As soon as all flows with a higher priority have been transmitted, a flow can be forwarded by a node. Each flow can be considered, at each instant, as more or less important to transmit compared to other flows in the node, depending on its priority value.
In our work, we suppose that the definition and specification of each message in the network, and the different criticality levels it belongs to, have already been defined. We consider message specifications and properties as static input informations in the network. We assume that it cannot be changed at runtime. That is also true for the criticality levels of a system, that we suppose to be statically-defined. It means that, during system utilization, we cannot dynamically add a new criticality level to the system or change the criticality levels a flow belongs to.
In safety-critical systems, the constraints applied to a system can correspond to specific critical situations. An airplane starting its landing, a spaceship during takeoff, a car during emergency braking phase are systems which will act out of their basical context. During these critical phases, the system needs more information acquisition, more frequently (more precise speed, more accurate position, more frequent fuel measurement, ...). As a conclusion, this critical behavior has to be properly defined, and we have to propose solutions to manage the system behavior during these phases.
During these critical phases, the flows which has been defined critical have to be guaranteed in their transmissions. It is not a matter of priority: the critical flows are not necessarily considered having higher priority than other non critical flows. It is the system designer who decides which flow is critical and which is not. It depends on what functionality has been determined as critical for the current phase. For example during the landing of an aiplane, landing gears deployment have to be not only assured, but also to be done in a finite delay even if there are competing flows. All the functions related to it will be designed as critical.
That concept goes beyond the notion of priority. Critical flows transmission must be assured, but also we must guarantee that non critical flows management will have a characterized and bounded impact on critical flows. That is what we call weak isolation constraints: the impact of non critical flows on critical flows is limited and can be computed. That leads to a question: how can we guarantee the scheduling and isolation of specific flows during critical phases of a system?

6.1.1 What is Mixed Criticality?

MC was first mentioned by its name in [114] and was presented for proccesor-oriented RT context. It consists in defining, implementing and managing different criticality levels inside RT systems. Current works on MC propose solutions to answer to these purposes in processor-oriented RT systems. Focusing on MC problems consists in solving design and scheduling problems due to the integration of different criticality levels inside the same system.
In order to satisfy the constraints of each critical level, MC works imply proposing delay computation models, scheduling techniques and criticality management protocols. RT scheduling integrating mixed critical approaches implies assuring that the timeliness, performance and isolation constraints due to each criticality level are satisfied.
In processor context, for each possible critical situation in a system, the system designer defines a specific set of tasks which have all to be guaranteed in their execution. It means that their worst case execution time has to be proved bounded. Each different set of tasks defined for a specific critical situation implies defining a different criticality level for the system. A mixed-critical system is a constrained system able to manage flows of different levels of criticality inside the same infrastructure: there is no physical isolation. This solution can be done by using weak isolation solutions: characterizing and bounding the impact of non critical flows on critical flows. The network can be dual-critical (Low (LO)-critical , High (HI)-critical) or it can integrate several different levels of criticality. A system submitted to MC levels integration is called multi-critical system.
MC has been introduced in RT systems with the objective of reducing costs, weight and energy consumption inside systems by mixing different functionalities of different criticality levels inside the same physical infrastructure.
Current criticality level is an information attached to a whole network. Each network can manage a certain number of criticality levels. Each flow in the network can belong to one or several of these criticality levels. The higher amount of criticality levels a flow belongs to, the more it would be a vital flow for the network. The number of criticality levels a flow can belong to depends on the number of criticality levels which had been defined by the system designer during the design phase. When a network is in a specific critical phase (landing mode for an airplane, for example), a specific criticality level corresponds to this phase. As a result, during this critical phase, we have to assure that all flows belonging to the current criticality level will be transmitted in a finite time.

6.2 Defining critical and non critical functions

Each flow transmission in the system will be defined for a certain certification level, depending of the criticality levels the flow belongs to. We have to compute the worst case end-to-end transmission delay of the flow in order to validate the weak isolation solution and, then, to meet certification requirements. It means that the more we define different criticality levels, the more we add constraints of certification to satisfy in order to prove the system reliability.
The integration of MC in RT networks implies defining a specific representation mode to illustrate the criticality levels of a network, and the different criticality levels a flow can belong to. For example in a personal car: all messages attached to Global Positioning System (GPS) tracking are critical for the driv-

ing control (mission-critical), but not in the case of crash avoidance or passenger protection which are situations where they are likely to be dropped. As the opposite, airbag triggering should be guaranteed in all the criticality levels. As a conclusion, each flow will belong to a various number of different criticality levels. Usually, we call a flow as ”non critical” when it only belongs to the lowest criticality level of the network.

6.3 Mixed criticality integration in networks

6.3.1 Isolation constraints

Each subnetwork is composed of different materials and is associated to a specific class of flows: one subnetwork for mechanical functions, one for comfort management, etc... This is a common solution for isolation: creating dedicated infrastructures for critical and non critical flows.
Dedicating a specific subnetwork for each class of flows allows us to guarantee a total isolation between flows which are not of the same criticality level. This assures a total isolation of each subnetwork: critical flows will not be blocked, cancelled or delayed because of non critical ones. This is a model we can find in several industrial architectures: for example, automotive constructors like BMW integrates different CAN buses in their vehicles in order to dedicate each one to specifc classes of flows (see figure 6.1).

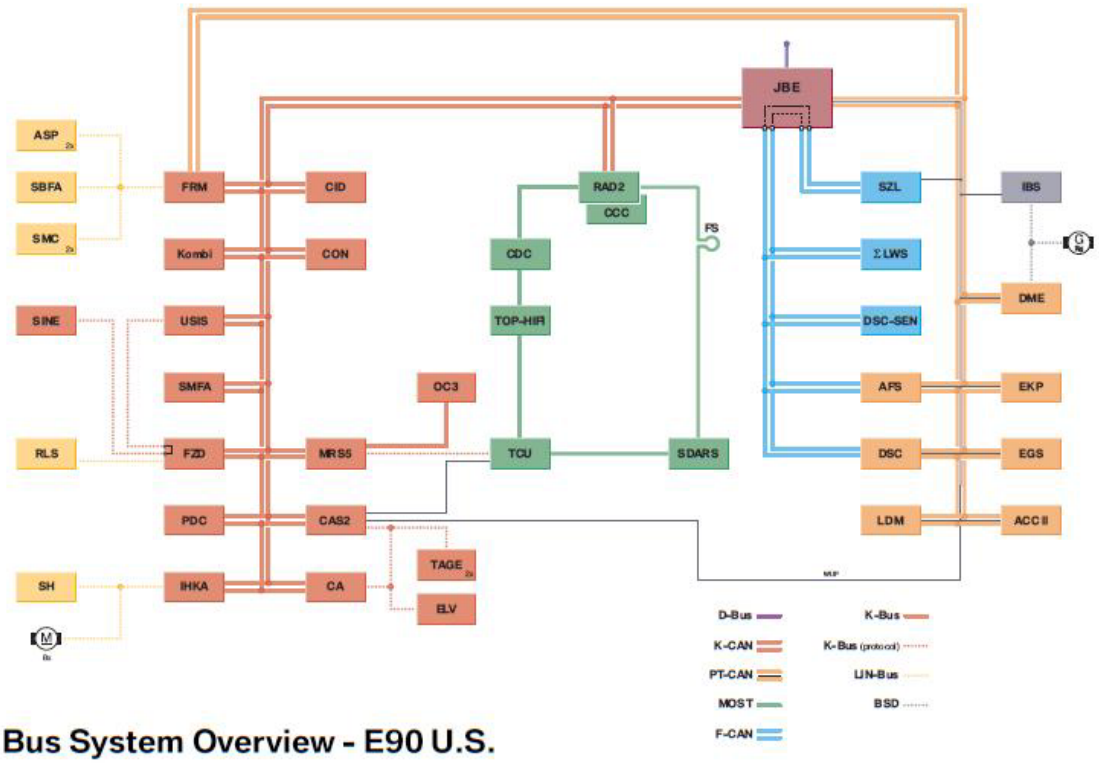


Figure 6.1: BMW mixed CAN buses

In the architecture presented in figure 6.1, we can distinguish dedicated communication buses for the different electronical commands embedded in a vehicle. We have functions such as EWS (electronical

start controller), doors blocking and CDC (Compact Disc Changer) associated to different CAN buses. Also, one bus is dedicated to car diagnosis and recovery for technical purposes. All this network architecture is centralized around the embedded computing unit present in the board computer, controlled by the driver.
This solution of physical isolation represents several problems. First, it has a strong financial impact. Multiplying the different network infrastructures implies multiplying the number of buses, wires, sensors and materials. It also implies building a new infrastructure for each class of needs and functions.
Secondly, having a dedicated subnetwork for each class of flows represents an increase in terms of weight and size. That implies designing the systems sufficiently big to find space to store each infrastructure. In embedded and mobility-oriented systems, each system is constrained in terms of size and weight, and certification authorities impose to satisfy specific constraints. As a conclusion, this process of just increasing the infrastructures sizes cannot be adopted.
Last but not least, multiplying the number of physical subsystems has a strong impact on energy efficiency. Each hardware has to be supplied, and as a result, the more materials we implement, the more energy ressources we need.
As a conclusion, we observe that increasing the size of a network cannot be presented as a global solution to timeliness, safety and performances problems. That is the purpose of MC: proposing a reliable solution to critical management while respecting ressources, size and energy constraints.

6.3.2 Mixed criticality for multicore platforms

In 2007, [114] presented MC in RT scheduling in processor context. It was described as a solution for deadline misses management and fault tolerance integration inside preemptive scheduling contexts for RT systems. At this time, MC was presented as a solution to certify task execution isolation. There has been several major works about MC in RT systems since then. We can mention [115] and [116] which were published in 2008.
The solution presented in these previous works, or in more recent works such as [117], consists in presenting MC integration in RT systems as a solution to classify tasks and manage the different criticality levels to implement in systems. Most of these works focused on new schedulability problems introduced by MC management. More recently, the work done in [118] concluded about the complexity of MC integration in RT systems. [119] focused also on the complexity represented by such new problems. This paper concluded that MC integration inside RT processor context, starting at 2 criticality levels, was NP-hard in the strong sense. This property represents one fundamental point of this work.
All the presented litterature, representing most of work done in MC domain, presents MC integration solutions in processor context. There has not been such a work today proposing MC modelling and integration inside RT networks. Nevertheless, some works like [120] proposed solution for criticality management and mode change integration inside RT systems. All these works consider MC level of a network as a static constant not likely to change. They do not focus on the potential dynamic evolution of the criticality level of a network, depending on its context of use. Eventually, the global shared hypothesis about MC in network context considers the criticality level as a static value in the network, which does not tend to change without specific intervention of the user or the system designer.
The purpose of our work is to propose solutions to extend MC integration to RT networks, first. We want

to propose a flow modelling integrating MC constraints, and to offer MC management protocols inside RT networks. Additionally, we want to focus on how to determine the criticality level of a network, and the conditions to modify it.

6.3.3 Weak isolation

In this work, we introduce the concept of weak temporal isolation between flows. It is defined by the logical isolation of critical flows from non-critical ones, without physical isolation. Flows from different levels of criticality will be temporally isolated but are sharing the same infrastructure. This weak isolation allows us to mix flows of different criticality levels on the same network while guaranteeing that the delay induced by non-critical flows transmission on critical flows transmisison will be bounded.

6.3.4 Mixed criticality in RT networks

Till now, there has not been any concrete implementation of MC management inside RT networks. Nevertheless, several solutions were proposed to integrate MC in specific contexts such as TTEthernet architectures [121], [122]. But these solutions only rely on costly clock-synchronized architectures. Similarly, a protocol was proposed in [123] to integrate MC inside Network-on-Chip (NoC) context. Nevertheless, this solution only considers dual-criticality level networks and do not focus on how to return to non criticality levels after a critical phase.
Another solution consists in isolating critical and non critical functions relies on creating dedicated subnets for each class of functions. This solution can be found, for example, in CAN and automation domain. Each dedicated subnet has its own bandwidth design. But, as in processor context, this method is very costful in terms of ressources. MC integration with weak isolation is the solution we propose.
Each function in a system corresponds a set of commands, lights, screens to accomplish it. These electronical and mechanical devices correspond to a various set of sensors, responsible for sending data flows corresponding to the function. As a result, each function of a system can be accomplished by a set of specific flows. Integrating MC management in RT network context allows us to mix all network flows inside the same infrastructure, in order to reduce the costs and ressources consumption of an embedded network. As all messages uses the same devices and topology, we can reduce the needs in terms of physical materials implementation.

6.4 Mixed criticality model representation

Our main goal in this work is to propose solutions to integrate and manage MC in RT networks. In order to do this, we need first to present the MC model we use along this work. In network context, we consider a network topology $\mathcal{N}$ composed of a set of flows $v_1, v_2, ..., v_n$. The network modelling is detailed in chapter 3.
All works about MC integration in RT systems consider the criticality level of a system as a global integer. This hypothesis has been detailed in such works like [117]. In order to represent MC inside a network topology $\mathcal{N}$, we first need to define the concept of a criticality level. A criticality level, denoted as γ, symbolizes one specific criticality level a RT network can switch to.

We suppose that the network is able to manage γ_{max} different criticality levels. If we note these different levels as $\{\gamma_1, ..., \gamma_{\gamma_{max}-1}, \gamma_{\gamma_{max}}\}$, we obtain: $\gamma_{max} = |\{\gamma_1, ..., \gamma_{\gamma_{max}-1}, \gamma_{\gamma_{max}}\}|$. The current criticality level of a network $\mathcal{N}$ is denoted as Γ and can change among a finite set of possible levels between γ_1 and $\gamma_{\gamma_{max}}$. For example in avionics or defense context, γ_{max} is usually contained between 2 (low, high) and 5 (non critical, performance-critical, mission-critical, vehicle-critical, safety-critical) [119].
In this work, we suppose that a switch has a unique MC level at any time. This level can dynamically change according to specific conditions (detailed further in this work). On the opposite, we consider that each network flow can belong to different criticality levels.
The complexity of the schedulability analysis of a RT network is directly linked to γ_{max}. Nevertheless, in order to be generic, we consider basically that a network can adopt any possible number of different criticality levels. We do not focus on the optimality of a scheduling algorithm in this work, but more on MC integration in networks. As a result, we can assume that γ_{max} is not limited by the model we propose. In this work, in the case of specific schedulability problems, our approach is to propose solution applicable to all situations whereas providing simple examples. That is why in this work, when applying our solutions to application examples, we suppose (without further details) that $\gamma_{max} = 2$ in order to limit the complexity of our implementations.

6.4.1 Two solutions for criticality modelling

In terms of flow model, integrating MC in network context implies to propose an improved model of flows. This can be done with two different potential modellings. We detail below these two different approaches, which can be used indifferently when focusing on MC integration inside RT networks.

Definition: WCAT-oriented model
A critical flow is modelled according to WCAT-oriented model when we consider that its WCAT increases with its criticality level.

Example

We suppose that a plane needs to land on a close airport after a travel. At the beginning, the pilot is initiating the landing phase, and the aircraft is starting to plummet progressively. As it is losing altitude and approaching the ground, the pilot needs more accurate information. It is obvious that making the difference between 20 and 50 feet altitude is more important in that case than differentiating 36000 and 36030 feet. That is why the different altitude sensors need to send more frequent messages about altitude information during landing. As the aircraft is approaching the ground, we need to measure its position more frequently to be prepared to the touch.
This example represents the first approach: increasing the frequency of messages transmissions in the case of higher criticality. This increase in the frequency of measures is represented by a decrease in the period (or minimum inter-arrival time) of a flow. This model is called a period-oriented criticality model. We consider in this model that, during critical phases, the period (or minimum inter-arrival time) of a flow will be modified. An end-system in our topology starts to transmit messages more frequently, with the same deadline. The altitude measurement by flow v_i can be represented as:

$$v_i = \{\vec{\mathcal{P}}_i, C_i, \vec{T}_i\}$$

with $\vec{T}_i = \{T_i^{\gamma_1}, T_i^{\gamma_2}, ..., T_i^{\gamma_{\gamma max}}\}$, the different minimum inter-arrival time depending on the different criticality levels $\gamma_1, \gamma_2, ..., \gamma_{\gamma_{max}}$.

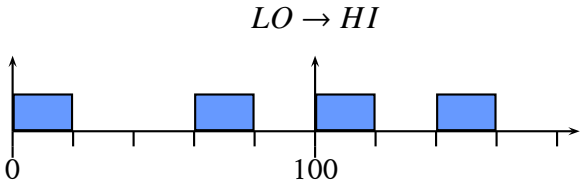


Figure 6.2: Period-oriented criticality management

In the case that the flow v_i does not belong to the criticality level γ_k, we set $T_i^{\gamma_k} = -1$.

Definition: Period-oriented model

A critical flow is modelled according to period-oriented model when we consider that its period decreases with its criticality level.

Example

The second solution for criticality level modelling is indexed on different WCAT for each criticality level. Let us come back to our example. During the same landing phase, the pilot will need more precise information in order to be more reactive and to have a more precise control over the plane. To answer to this need, the sensors (the speed detection one for example) will start to send more precise information (500.32 mph instead of 500 mph). It means that the sensors will start to send messages containing more information. As a conclusion the size of the message will increase (32 bytes for a int compared to 64 bytes for double).
This representation is modeled by attributing a specific size to the flow for each possible criticality level. We know that the WCAT and the size of a flow are related through the bandwidth of the network. As a conclusion, this second approach consists in attributing a specific WCAT to the flow for each criticality level. This method is called the WCAT-oriented method. The corresponding flow v_i representation is as follows:

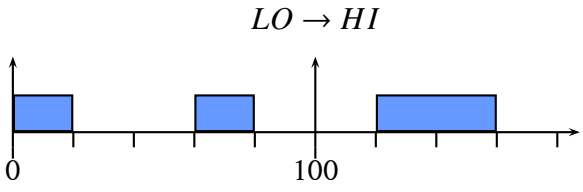


Figure 6.3: WCAT-oriented MC management

$$v_i = \{\vec{\mathcal{P}}_i, \vec{C}_i, T_i\}$$

with $\vec{C}_i = \{C_i^{\gamma_1}, C_i^{\gamma_2}, ..., C_i^{\gamma_{\gamma max}}\}$. This is the WCAT-oriented hypothesis: in critical situations, frames can contain longer messages than in non critical situatons and represent a longer WCAT.

In the case that v_i does not belong to the criticality level γ_k, we note $C_i^{\gamma_k} = -1$.
These two approaches can be used indifferently in order to represent MC integration inside RT networks. They correspond to a different flow model, but they correspond to the same implementation. In terms of

MC integration, we represent in this work the representation with both models. In previous MC works such as [114], the default used model is the WCAT-oriented one.

6.4.2 Mixed criticality in nodes

Each flow has a specific WCAT for each criticality level. For the criticality levels it does not belong to, this WCAT is equal to -1. In a related work [124], we proposed a solution to tag each Ethernet message with its highest criticality level. This work is detailed in annexe (see section 14.1).
In order to manage MC, we define the criticality level of a switch. It corresponds to the criticality level of messages a switch is authorized to transmit. If a message criticality level tag is lower than the criticality level of a switch in its path, this switch will drop out the message.

Definition: Switch criticality level
The criticality level of a switch corresponds to the minimum required criticality level for a message to be transmitted by this switch.

6.4.3 The hierarchical hypothesis

In RT networks, infrastructures can have to model more than 2 different criticality levels. For example in a vehicle, it is common to integrate a specific criticality level for all functions dedicated to mission or process execution (mission-critical), functions responsible for the physical integrity of the system (integrity-critical) and finally only functions to assure the safety of its occupants (safety-critical). We represent the number of criticality levels of a system as γ_{max}
Our work is based on the following hypothesis: no matter the number of criticality levels managed by a system, each level can be defined as 'more' or 'less' critical than another one. Assuring the mission-critical messages transmission is more critical than assuring the transmission of non critical messages but it is less critical for the network than assuring the safety of vehicle's occupants.
Based on this observation, we extracted the following hypothesis: all criticality levels in a network can be organized following a hierarchical structure. In his work, Vestal [114] formulated this hypothesis in processor context by supposing that the 'more' critical a level, the higher the level of certification it has to be compliant to. This has various implications in our work.
First, it means that the estimation of a flow WCAT is more and more pessimistic with the increase of the hierarchical importance of a criticality level. This hypothesis about the hierarchical structure among all criticality levels inside a network represents the fundamentals of MC modelling in our work, and we propose to expose here its application to RT network context.
In terms of modelling, the Vestal hypothesis (in processor context) supposes a system composed of γ_{max} levels of criticality, denoted as $\{\gamma_1, ..., \gamma_{\gamma_{max}-1}, \gamma_{\gamma_{max}}\}$. If we suppose a system composed of a set of n tasks $\tau_1, ...\tau_{n-1}, \tau_n$, each task τ_i can be represented by a 2-tuple $\{\vec{C}_i, T_i\}$, with $\vec{C}_i = \{C_i^1, C_i^2, ..., C_i^{\gamma_{max}}\}$ (WCAT-oriented model). Given the Vestal hierarchical hypothesis, we can make this assumption: if γ_a is less critical than γ_b, we note $\gamma_a < \gamma_b$. It that case, C_i^a will be shorter than C_i^b for all tasks τ_i in the system. The application of this hypothesis gives us (in the case where $C_i^{\gamma_a} \neq -1$ and $C_i^{\gamma_b} \neq -1$):

$$\forall a, b \in [1, ..., \gamma_{max} - 1, \gamma_{max}], \gamma_a < \gamma_b \implies \forall i \in [1, n], C_i^{\gamma_a} \leq C_i^{\gamma_b} \quad (6.1)$$

If we want to apply to network context, we suppose a network $\mathcal{N}$ composed of n flows $\{v_1, ..., v_{n-1}, v_n\}$ and γ_{max} criticality levels $\{\gamma_1, ..., \gamma_{\gamma_{max}-1}, \gamma_{\gamma_{max}}\}$. We represent each flow v_i as a 3-tuple $\{\vec{\mathcal{P}}_i, \vec{C}_i, T_i\}$ (WCAT-oriented model). We obtain the exactly same hypothesis for WCAT-oriented network modelling (in the case where $C_i^{\gamma_a} \neq -1$ and $C_i^{\gamma_b} \neq -1$):

$$\forall a, b \in [1, ..., \gamma_{max} - 1, \gamma_{max}], \gamma_a < \gamma_b \implies \forall v_i \in [v_1, ..., v_{n-1}, v_n], C_i^{\gamma_a} \leq C_i^{\gamma_b} \quad (6.2)$$

This hypothesis is a fundamental assumption in our work for MC modelling and integration. Considering the actual representations of criticality inside RT industrial networks, this hypothesis is representative of real implementations. Each class of messages (in independant subnets) is considered as more or less important to others. Thus, modelling criticality without integrating this hypothesis can be considered as equivalent to a classification problem, answering to the following question: which flow belongs to which level? Answering this question is not the purpose of this work. In our work, we suppose that determining the criticality level of each flow is done at network design phase (see section 6.2).
In our modelling, the direct application of the Vestal's hypothesis is only partially complete. The proposed hypothesis only considers WCAT-oriented modelling and does not include the period-oriented modelling we introduced. Considering a network $\mathcal{N}$ composed of a set of flows $\{v_1, v_2, ..., v_n\}$, each flow v_i chararacterized by $v_i = \{\vec{\mathcal{P}}_i, C_i, \vec{T}_i\}$. We suppose that the more critical a criticality level, the shorter the period of each message (defined for this level). We can represent Vestal's hypothesis applied to period-oriented modelling as:

$$\forall \gamma_a, \gamma_b \in [1, ..., \gamma_{max} - 1, \gamma_{max}], a < b \implies \forall i \in [1, m], T_i^{\gamma_a} \geq T_i^{\gamma_b} \quad (6.3)$$

Considering the WCAT-oriented and period-oriented approaches in MC modelling, the hypothesis of Vestal allows us to build a hierarchical structure among each criticality level $\gamma_1, ..., \gamma_{\gamma_{max}-1}, \gamma_{\gamma_{max}}$ in the network. We adapted the hypothesis of Vestal to RT networks context by combining the two previously made assumptions. We obtain the following result:

$$\forall a, b \in [1, ..., \gamma_{max} - 1, \gamma_{max}], a < b \implies \forall v_i \in \mathcal{N}, \begin{cases} C_i^{\gamma_a} \leq C_i^{\gamma_b} \\ T_i^{\gamma_a} \geq T_i^{\gamma_b} \end{cases} \quad (6.4)$$

This hypothesis means that, for every criticality levels γ_a and γ_b in the network, as soon as γ_b is consid-

ered as more critical than γ_a, all flows from γ_b have necessarily a longest WCAT and a shorter period (or equal). This leads to consider of a hierarchical organization of the criticality levels inside a network.

6.4.4 Criticality level assignment

In the following work, when there is no specific hypothesis, we consider that we are working with the WCAT-oriented approach. We make the hypothesis that each message is defined with a dedicated WCAT for each $\gamma_1, \gamma_2, ..., \gamma_m$ criticality level it belongs to.
Considering the hypothesis we formulated previously on the hierarchical structure of MC levels, we deduce another assumption from it. In criticality levels design and modelling, we consider that if a flow v_i is a defined for a criticality level γ_m with $C_i^{\gamma_m} \neq -1$, then the flow will also belong to all criticality levels $\gamma_1, ..., \gamma_{m-2}, \gamma_{m-1}$. It means that, if a flow is defined for a specific criticality level, it also belongs to all the lower criticality levels.
Concretely, this hypothesis makes sense. We consider that if a message is considered as critical for the safety of the occupants of a vehicle, the message is also critical for the structural integrity of the vehicle (if we consider the vehicle-critical level as lower than the safety-critical level). This assumption is issued from the hierarchical hypothesis we formulated previously.
In order to focus on the impact of this assumption in criticality level management, we show it on a dual-criticality level network example. This example is detailed below.

Application to a dual-criticality level network

We want to show on a dual-criticality level network that, in worst case analysis, if a flow is defined for a criticality level, it also belongs to all lower criticality levels. In order to do this, we suppose a network $\mathcal{N}$ composed of a set of n flows $\mathcal{V} = \{v_1, ..., v_{n-1}, v_n\}$. We suppose the network as composed of a set of 2 criticality levels LO and HI. We consider that all flows from v_1 to v_k are LO-critical flows ($C_i^{HI} = -1$) and, on the opposite, we suppose that all flows from v_{k+1} to v_n are HI-critical flows ($C_i^{HI} \geq C_i^{LO} > 0$).
We start by supposing that flows from HI level $(v_{k+1}, ..., v_{n-1}, v_n)$ only belong to HI level. It means that we consider $\forall v_j \in HI, C_j^{LO} = -1$. To measure the impact in the network, we compute the respective LO u_{LO}^{LO} and HI u_{HI}^{HI} network utilizations represented by the flows in the network. We express their value as follows:

$$u_{LO}^{LO} = \sum_{v_i \notin HI} \left(\frac{C_i^{LO}}{T_i}\right)$$
$$u_{HI}^{HI} = \sum_{v_i \in HI} \left(\frac{C_i^{HI}}{T_i}\right) \quad (6.5)$$

We define LO u_{LO}^{HI} and HI u^{HI} loads, respectively corresponding to :

$$
\begin{aligned}
u_{HI}^{LO} &= \sum_{v_i \in \gamma_{HI}} \left(\frac{C_i^{LO}}{T_i}\right) \\
u_{LO} &= u_{LO}^{HI} + u_{LO}^{LO} \\
u_{LO} &= \sum_{v_i \in \gamma_{HI}} \left(\frac{C_i^{LO}}{T_i}\right) + \sum_{v_i \notin HI} \left(\frac{C_i^{LO}}{T_i}\right)
\end{aligned}
\tag{6.6}
$$

Considering our hypothesis, we have $u_{HI}^{LO} = 0$ and $u_{LO} = \sum_{v_i \notin HI} \left(\frac{C_i^{LO}}{T_i}\right)$.
In a second time, we consider the opposite hypothesis : we suppose that all messages from HI level have also a dedicated WCAT in LO level. It means that all flows $\{v_1, ..., v_{n-1}, v_n\}$ verify the property $C_i^{LO} \neq -1$. For an identical network static definition of flows, the value of u_{HI}^{HI} and u_{LO}^{HI} are strictly the same, which mean that the assumption we made has no impact on HI flows traffic computation. On the contrary, for LO-critical flows, we have

$$
u_{LO} - u_{LO}^{LO} = u_{HI}^{LO} = \sum_{v_i \in \gamma_{HI}} \left(\frac{C_i^{LO}}{T_i}\right) \geq 0
\tag{6.7}
$$

As a result, we obtain that $u_{HI}^{LO} \geq 0$ and $u_{LO} \geq \sum_{v_i \notin HI} \left(\frac{C_i^{LO}}{T_i}\right)$. It means that our second hypothesis provides solutions including the result of the first case. It means that supposing that all flows from a level also belong to less critical levels is an hypothesis which is more pessimistic. The worst case analysis of these situations includes situations where there is no LO-WCAT for HI messages.
We generalize this to a system of γ_{max} different criticality levels by computing the dedicated load u_{γ_k} for each criticality level γ_k in the network, and comparing it with $u_{\gamma_k}^{\gamma_k}$. It means that, in worst case analysis, considering that critical flows also belong to lower criticality levels represents a higher amount of low critical traffic and a more pessimistic analysis. As a result, we assume as true the following assumption :

$$
\forall k \in [1; \gamma_{max}], C_i^{\gamma_k} > 0 \implies (\forall j \in [1; k-1], C_i^{\gamma j} \geq 0)
\tag{6.8}
$$

6.5 Mixed criticality implementations and protocols

The goal of MC management protocols is to provide solutions to verify and change the criticality level of a network, depending on the topology and flow constraints. In processor context concerned by RT scheduling, criticality is supposed as a global information which can be changed instantly in the system, with no memory ressources nor time costs. It means that MC integration in processor context implies that criticality management by itself has no impact on scheduling or network performances and relia-

bility. This is the hypothesis we can find in works like [117], [125].
In network context, this hypothesis is not relevant anymore. We cannot assume that changing and synchronizing the criticality level of all nodes in a topology does not cost time. Transmitting a message containing criticality management informations from one node to another takes a certain delay, due to switching latency, WCAT management and network traffic additional induced delay. That is why we have to propose, based on the MC integration model we detailed in this chapter, solutions to integrate MC management inside RT networks. This solutions will have to take into account the particular constraints and context of use due to network design (distributed context architectural limits).
It means that, in our work, we have to provide solutions to estimate the different delays induced by MC management inside a network. In order to be compliant with our certification purposes, these delay would have to be computed in a worst case analysis approach (detailed in chapter 7 and 8).

6.6 Conclusion

Proposing integration of MC in network context has not been much investigated in the state of the art . The purpose of this work is precisely to focus on MC modelling and integration inside RT networks. In order to answer to this problematic, we propose to detail in the following chapters different protocols for MC integration inside Ethernet networks.
We already proposed in previous work [126] a potential concrete implementation of MC management inside Ethernet. We detail this integration protocol in the next chapter.

Chapter 7

Centralized MC management

"Nous ne vivons que pour trouver la beauté. Tout le reste n'est qu'attente."

"We only live to discover beauty. All else is a form of waiting."

– Kahlil Gibran [127]

Contents

7.1 Introduction

7.1.1 Mixed criticality integration

Context

In [126], we describe a concrete implementation of MC management inside Ethernet. The following work is a detailed version of what was presented.
The needs for comfort and reliability in such domains like automation, avionics and public transports (see section 2.1.3) are increasing. This induces an increase in the number of functionalities to manage at the same time, in embedded networks. In these industrial domains concerned by RT, we have to manage different type of messages coming from subnetworks: mechanical commands (brakes, acceleration, wheel control), displaying driving information (current speed, GPS trajectory, time and date), comfort management (air conditioner, music and multimedia players).
In classical RT and embedded networks each class of functions is treated in a separate infrastructure in order to guarantee the isolation between messages of different criticality. In that case, each independant subnetwork relies on a specific physical infrastrure, designed with its own constraints in terms of bandwidth and materials.
This solution implies increasing costs as we need to implement dedicated infrastructures. Thus, this solution also implies a loss of space and energy consumption: each independant infrastructure occupies its own space, plus the eventual gaps to manage heat dissipation between materials and minimum distance to avoid electrical interferencies. The solution of dedicating a specific network infrastructure for each class of function can quickly appear to be costful and complicated to manage. That is why we propose to implement MC management inside RT networks.
See chapter 6, MC was already been introduced in processor context to answer these problems [119]. We want to propose MC management for RT networks. But this implies defining a specific protocol to manage MC. In the following chapter, we detail such a protocol.

Modelling

We make the assumption (see chapter 6) that, in a common infrastucture, critical and non critical flows cannot be transmitted simultaneously without network overload. But certification purposes implies guaranteeing no overload.
We want to implement weak isolation with MC integration, to provide a solution to guarantee that critical flows will not be delayed or bothered by non critical flows. It seems obvious to suppose that, when there is no overload in network traffic, not any message in the network is delayed because of another message transmission. The integration of MC was proposed as a solution to manage these overloads in RT networks, in order to guarantee critical messages transmissions even during critical phases. It means that MC integration has to guarantee critical flows scheduling. That leads to the following question: how do we assure critical messages transmission and avoid network overloads?
An overload in node from network $\mathcal{N}$ is characterized by the instant when the input traffic rate of a node is too high to manage all incoming messages with the common bandwidth. An overload can be

characterized by computing the utilization of messages in a node. For a given criticality level γ, the utilization represented by messages of γ level in node s, is represented by the load of messages from which γ are their highest possible criticality level. We note this utilization as $u^{\gamma}(s)$. If we suppose that $\gamma < \gamma_{max} - 1$, we obtain:

$$u_{\gamma}^{\gamma}(s) = \sum_{\substack{v_i \in \gamma \\ v_i \notin \gamma+1}} \left(\frac{C_i^{\gamma}}{T_i}\right) \tag{7.1}$$

If we suppose a network composed of $\gamma_1, ..., \gamma_{\gamma_{max}-1}, \gamma_{\gamma_{max}}$ different levels of criticality, we characterize an overload in node s by the following condition:

$$\sum_{j=1}^{\gamma_{max}} u_{\gamma_j}^{\gamma_j} > 1 \tag{7.2}$$

If we apply the following condition on a dual criticality level network, composed of LO and HI levels, we obtain the following condition: $u^{HI}(s) + u^{LO}(s) > 1$. That gives us, in a specific node s: $u_{HI}^{HI} + u_{LO}^{LO} > 1$. As long as network traffic stays manageable in all the nodes i of network $\mathcal{N}$ ($\sum_{j=1}^{k} (u_i^{\gamma_j}) < 1$), we can assure that any message will be transmitted in a finite time. This delay can be computed with the Trajectory Approach. This delay depends on the different waiting delay induced by the transmission of other messages. As long as the nodes queue size decreases with time, we can assure that there is a time instant when our message will be necessarily transmitted.

The main goal of MC management is to assure the transmission of all critical messages, especially during critical phases triggered by a change to a higher criticality level. As an hypothesis of this work, we conside that the LO-critical messages (messages belonging to the lowest criticality level) do not have any constraint in terms of transmission time. Messages from LO level can be managed with best effort, or dropped out during critical phases without impact (except from potential non-preemptive effect) on the safety or reliability of the network.

The ability to send all the critical messages implies a second constraint in the network. For each criticality level γ, we must verify $u_i^{\gamma} \leq 1$. It means that the set of only γ-critical messages should be manageable by itself in terms of traffic. If this condition is not verified, it means that there is a conflict between several messages of the same criticality level, which is a design problem.

Problem statement

We want to focus on the importance of MC management inside systems. We focus on the basic example of a public transport bus. We suppose that we have different subnetworks connected to the same final switch (see figure 7.1). We represent this network traffic with flow modelling issued from the Trajectory Approach. In this network, the first subnetwork concerns mechanical functions of the vehicle,

which have to be transmitted and are considered as critical. The two other subnetworks are for ticket management and passenger information, and are not considered as critical for the vehicle.

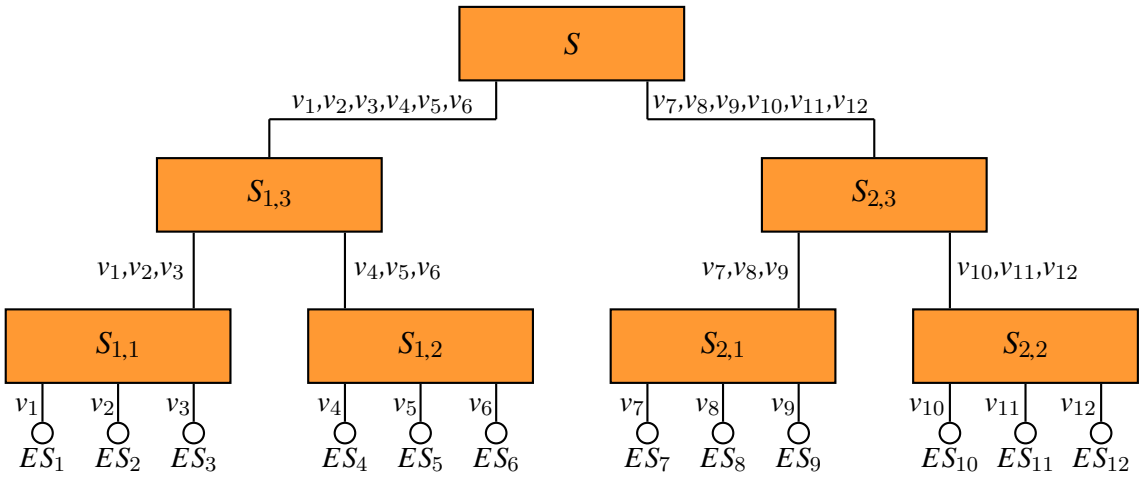


Figure 7.1: Subnetworks connection

In our example, we suppose a dual-criticality level network, composed of LO and HI levels. Starting from this description of the flows, we can deduce the LO and HI utilization rates for each switch, with the following expression in switch S: $u_{LO} = u_{LO}^{LO} + u_{LO}^{HI}$, $u_{HI} = u_{HI}^{LO} + u_{HI}^{HI}$. We suppose the different flows characterized by the following parameters:

Flow	$C_i^{LO}(\mu s)$	$C_i^{HI}(\mu s)$	$T_i(\mu s)$	u_{LO}	u_{HI}
v_1	10	20	100	0.1	0.2
v_2	20	-	200	0.1	-
v_3	10	-	100	0.1	-
v_4	10	20	100	0.2	0.2
v_5	10	20	200	0.05	0.1
v_6	50	-	500	0.1	-
v_7	10	-	200	0.05	-
v_8	20	-	400	0.05	-
v_9	20	40	400	0.05	0.1
v_{10}	10	20	200	0.05	0.1
v_{11}	20	-	400	0.05	-
v_{12}	20	50	200	0.1	0.25

Switch	u_{LO}	u_{HI}	$u_{LO}^{LO} + u_{HI}$
$S_{1,1}$	0.3	0.2	0.4
$S_{1,2}$	0.35	0.3	0.4
$S_{1,3}$	0.65	0.5	0.8
$S_{2,1}$	0.15	0.1	0.2
$S_{2,2}$	0.2	0.35	0.55
$S_{2,3}$	0.35	0.45	0.75
S	1.0	0.95	1.55

As we can see, the combined LO and HI utilizations of LO messages in HI messages in HI mode ($u_{LO}^{LO} + u_{HI}$) exceed the maximum possible utilization in switch S. This prevents the messages from being schedulable in a finite amount of time ($u_{LO} + u_{HI} > 1$).

During HI-critical phases, we cannot guarantee both the transmissions of LO and HI-critical messages. In standard Ethernet without MC management, this is a situation of overload. This overload represents a problem : potentially, HI-critical flows in the network will not be transmitted on-time because of a non-bounded induced delay due to LO-critical flows transmission. It means that the weak temporal isolation will not be respected.

To solve this problem, we want to implement solutions to guarantee HI critical traffic transmission. That implies, if we want to use a global topology, implementing a protocol to manage MC different criticality levels. With this protocol, we will be able to guarantee HI critical messages transmission inside the topology.

7.1.2 Requirements

Central node

In processor-context, for integrating MC management among scheduling models, we make the hypothesis that the criticality information is a global information suitable to all cores [117], [125]. It integrates a global and centralized management of the criticality information, which is considered as common all over the system. In network context, this hypothesis of centralization supposes a consistant criticality level management among all nodes in the network. We have to consider criticality changes and synchronization costs in our approach.
The objective of the MC management protocol that we present is to reproduce MC processor-oriented management in network context. We propose here to build a MC management protocol based on a central node for managing the criticality level. As we are in network context, we need to precisely define how to share the criticality level information among all the network.
In order to integrate consistency in criticality level management for all switches of the network, we propose to introduce the concept of a central node of the network. Usually, the different network topologies are organized conformly to a specific network topology: a tree-oriented topology, composed of interconnected nodes with no loop. That is the case of such industrial networks such as AFDX, CAN for public vehicles, for example, which are organized based on a tree structure.
The network topologies we based our work on are organized following this rule: the topology should have a node which could be considered as a "central" node. This central node is defined by the network designer. Additionnally, as we base our work on the Trajectory Approach, we consider that the networks we focus on do not contain any loop. This implies the following conditions:

- Each path between two nodes is unique. There is no possibility of building several different paths between two nodes in the network.
- You cannot build a path from a node to itself without using at least twice the same link.

Depending on the topology shape (tree or star for example) the choice of the central node can be either arbitrarily designed, or built according to a specific logic. The central node in the network has a certain number of intermediate nodes between itself and any node in the network. By scanning each node in the network, we keep the highest number of intermediate nodes we found for a given node, which corresponds to the distance between the central node and its farest node. As shown later (see 7.2.4) the delay to change the criticality level increases as a function of this distance. It means that the choice of the central node should be done in order to minimize this value.
The central node will be responsible for storing the criticality current value of the network in its internal memory, and keeping the consistency of criticality level in the network. The protocol presented below is based on the approach of defining this central node.

Clock synchronization

We are working in RT network context. Transmission delays must be bounded inside the network, and that includes delay needed to change the criticality level of the network. Consistency and determinism constraints on the transmission delay implies having a global time reference among all the nodes in the network. This global time can be assured only with clock synchronization. As a result, our MC management protocol implies implementing clock synchronization in the network. All the nodes should integrate a clock synchronization protocol, and the devices used as switches should be compliant with this protocol. In order to rely on a standard synchronization protocol, we suppose that the network is composed of IEEE-1588 switches, integrating PTP (see section 3.3 on part I).
In terms of modelling, we suppose that the network $\mathcal{N}$ is composed of a set of switches $\{S_1, S_2, ..., S_o\}$, all defined as PTP compliant. Any switch S_n has an internal clock $Cl(S_n)$. We define a global clock in the network, which will be used as a reference. PTP offers a poll-based algorithm to define the reference clock in the network. For more details about the clock reference definition, see details in the PTP specification [48].
If we suppose that the global clock denoted as Cl_c, it means that each clock has its own clock jitter $J_\epsilon^{S_n}$ computed with:

$$J_\epsilon^{S_n} = \mid (Cl(S_n) - Cl_c) \mid \tag{7.3}$$

We want to evaluate the worst case transmission delays for the protocol integration, meaning that we consider only the worst jitter in the network. We note as follows the clock accuracy in the network:

$$\epsilon = \max_{S_n \in \mathcal{N}} (J_\epsilon^{S_n}) \tag{7.4}$$

We use this expression of ϵ in all the following work. This expression allows us to provide a worst case delay analysis while taking into account all potential clock accuracies in the network. This allows us to assure the determinism of transmissions in the network, required by certification authorities.

7.2 A TWO-PHASE PROTOCOL

We define a MC management protocol for RT networks. This protocol is based on a centralized approach of MC management. The centralized MC-management protocol is a network protocol whose role is to assure the transmission and consistency of the criticality level and criticality changes informations among a dedicated network topology. This protocol is in charge of maintaining the same criticality level in every node in the network. In order to do this, the protocol is designed to share criticality information among all the nodes in case of a need to increase the current level. The working mode of this protocol to change the current criticality level of the network is based on two different phases. We detail below each one of these phases.

7.2.1 Call phase

The first phase of the protocol is the call phase. We consider a network $\mathcal{N}$ in a current criticality level γ_{anc}. The call phase includes the detection of a need to change the criticality level, and the transmission of this information to the central node of the network. First, we detail the conditions to trigger a criticality level change. The detection of a need to change the criticality level can happen in any switch of the network. This detection is based on two different events: detecting a message WCAT overrun (or higher criticality level) and deciding if the switch needs to call for a criticality level change or not.
Detecting a message exception consists in detecting that a received message in the node corresponds to a different criticality level than the current one. This can correspond to two situations:

- Either a message of a given flow starts to have a longer WCAT than its γ_{anc}-WCAT (see figure 6.3).
- Or the flow starts to produce messages with a shorter period (or inter-arrival time) (see figure 6.2).

If one of these events is detected, it means that the node received a message which corresponds to a higher criticality level than the current one and needs to call for a change. The current node is called the triggering node.
The criticality level the node wants to change to is computed according to the transmission delay of the exception message. This computation is based on a threshold mechanism. The switch detects the transmission delay C of this message and we compute the WCAT value corresponding to the closest threshold. Corresponding to this WCAT, we deduce the criticality level to change to.
In other words, if we suppose a message m characterized by a vector of WCAT $\{C_m^{\gamma_1}, ..., C_m^{\gamma_{k-1}}, C_m^{\gamma_k}\}$, we search for the smallest value of u verifying $C_m^{\gamma_u} \geq C$. This supposes that all nodes in the path of the flow are aware of the flow properties. This implies configuring the different nodes of the network.
Once we detect WCAT overrun or shorter period, this triggers the transmission of a message from the current node to the central node. This message is named a Switch-Criticality Call (SCC), and has a WCAT denoted as C_c. SCC message contains the computed value γ_j of the criticality level the current node calls to change to. The detection of the event and the transmission of the SCC message corresponds to the call phase.
The SCC message is transmitted to the central node. As it is a message dedicated for configuration and important for transmission of critical messages, we attribute it the highest possible priority (dedicated for configuration messages like PTP messages) according to Ethernet 802.1Q protocol. In switched Ethernet context, applying a fixed-priority to a message is possible according to a structure of VLAN. Each physical link between two nodes is splitted into several different VLAN (max 4096), and each VLAN is associated to a given priority. In each node, then, each incoming message on a given VLAN is queued up according to the priority of the VLAN it belongs to. We transmit the SCC message to the VLAN associated with the highest priority (see figure 7.2).

7.2.2 Multicast phase

Once the central node gets the SCC message, it gets the new criticality level γ_{new} detected by the triggering node. The central node has then to decide whether or not to change the current criticality level of the network. This decision is based on several hypothese made based on the criticality level organization.

MC has been introduced in RT domain to define classes of messages which have to be assured. We already discussed in section 6.4.3 about the hierarchical structure of criticality levels. For example, the safety of occupants is more critical than safety of the mission, which is more critical than safety of the vehicle, etc... In this centralized MC management protocol, we reuse this hypothesis. No matter the number k of criticality levels, we can define a hierarchical order of importance between the criticality levels of a network. The more critical a level, the higher the level of requirements.
In case that a flow is not defined for a specific criticality level, then we denote its WCAT as equal to -1, and same for its period. Based on this hypothesis (detailed in [117]) we can establish an algorithm to determine whether or not we need to order criticality level change in the network. We can distinguish two situations:

In the first situation, the new level γ_{new} is higher to the ancient level γ_{anc} ($\gamma_{new} > \gamma_{anc}$), meaning that the new level is more critical than the previous one. In that case the central node immediately orders a criticality level change to all nodes in the network. This context corresponds changing, for example, from LO to HI for a two criticality level network (see figure 7.2).

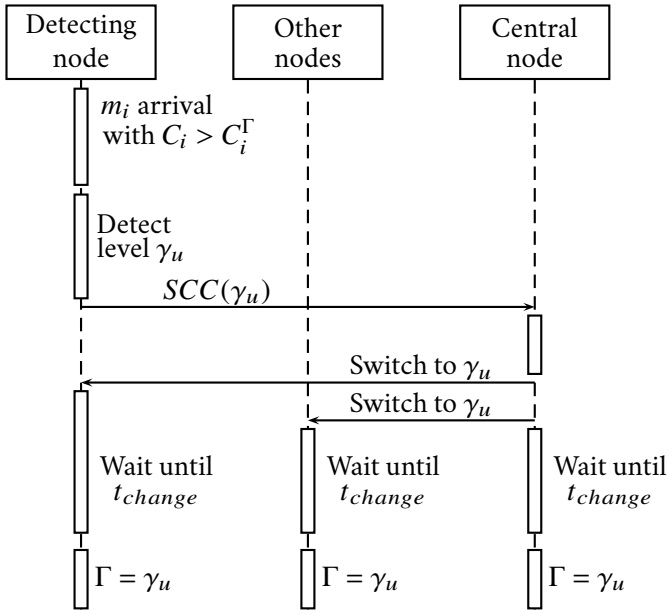


Figure 7.2: Increase criticality level

In the second situation, he new level γ_{new} is lower to the ancient level γ_{anc}. This corresponds to a need to decrease the criticality level, and we detail this specific case in section 7.2.3.
In both cases, the criticality level change relies on a reliable multicast protocol, allowing the central node to transmit the same message to all nodes in the topology. We need to prove the reliability of this multicast.

Total order

The MC management protocol we present here is based on the centralization of criticality level information. We define a global criticality level (stored in the central node) and we propose solutions to share and modify it simultaneously (with an accuracy of ϵ) on all nodes of the topology. To assure coherency in the criticality level management, we need to assure the total order [128] of criticality orders management.
Total order in an embedded or distributed system consists in assuring that a sequential set of actions will be executed in the same order in all the nodes of a network. If we suppose a piece of information distributed on several nodes (for redundance purposes, for example), the different actions likely to modify

this piece of information must be executed in the same order in all nodes, in order to guarantee the consistency of information in the network.
We illustrate this phenomenon through a simple example. We suppose three switches: $\{S, S_1, S_2\}$, organized according to the topology described in figure 7.3. We suppose that S_1 and S_2 store the value of a numeric variable, denoted as a. Basically, we note the value of a in S_1 as S_1^a and the value of a in S_2 as S_2^a.
At two different dates t_1 and t_2, S transmits the following actions: A and B. As shown in figure 7.4, if there is no assumption about total order in the network, there is no guarantee that actions A and B will be executed in the same order in S_1 and S_2. As a result, this can lead in obtaining different values of a in the nodes S_1 and S_2, breaking the consistency constraint of the network.

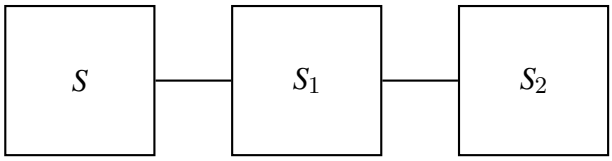


Figure 7.3: Example topology

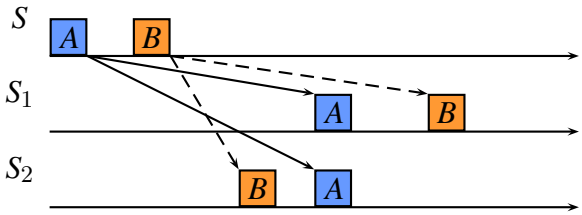


Figure 7.4: Scheduling without total order

Assuring the total order of criticality change upate in a network consists precisely in assuring that successive criticality level changes will be executed in the same order in all nodes of the network. That is a constraint we have to respect in our MC management approach. In order to assure the respect of this constraint, we provide a solution guaranteeing that actions A and B will be executed at the same instant in all the nodes. To implement this in MC centralized management protocol, we propose a reliable multicast protocol to share criticality level information in the topology.

Reliable multicast

We suppose that there is a need to change the criticality level. Our goal with the centralized MC management protocol is to assure the transmission of the new criticality level information γ_{new} to all nodes in the topology, in a bounded delay and preserving the consistency of the criticality level. Thus, in order to preserve the consistency of the current criticality level in all the nodes, we need to guarantee a total order in the criticality updates: two consecutive criticality changes have to be executed in the same order in all the nodes.
A reliable real-time multicast is a method to send the same information to all nodes in a network, providing total order for the update of the criticality level in all nodes. In [129], the authors show how to build a real-time reliable multicast, provided that worst case messages end-to-end transmission delays can be upper bounded. Based on this proposition, we built a solution adapted to the context of MC management.
As we are working on deterministic networks, each worst case end-to-end transmission delay in the network is bounded. In order to compute this upper bound for each message from each flow, we based our work on the Trajectory Approach. Since we can assure a bounded transmission delay for MC information in each physical link of the network, we can provide guarantees on transmission delays in the whole

network.
Figure 7.5 presents the MC centralized protocol criticality level change process. We observe the two phases of the mode change and their corresponding configuration message (SCC, multicast).

First, we have the calling phase during which there is a transmission of an SCC message from the triggering node to the central node. In a second step, the central node multicasts the new criticality level to all the nodes in the network, in the case where the criticality level has increased.

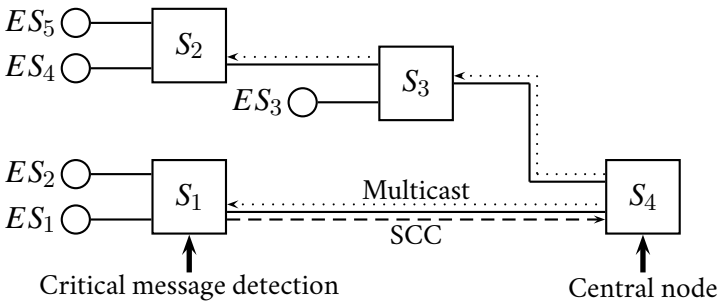


Figure 7.5: MC management centralized protocol

Once a node gets the multicast message, it does not change its criticality level instantly. We have to assure that the criticality level stays the same in each node of the network, at each instant (with an accuracy equal to the clock accuracy ϵ). It means that the criticality level change happens at the same date t_{change} in all the nodes. This can happen only when the last node received the multicast message. At this moment, we change the criticality level to γ_{new} in all nodes. Considering the potential clock accuracy, it means that we can guarantee that, at a date $t_{change} + \epsilon$, all the nodes in the network will have increased their criticality level to γ_{new}.
We want to illustrate this through an example, in a dual-criticality level network (LO, HI). If we focus on the example topology described in figure 7.5, we obtain the scenario described in figure 7.6.

The multicast message m is emitted from central node S_4. It is received and analyzed at different dates by each node. As a result, each node will be ready at a different date (respectively t_1 for S_1, t_2 for S_2, t_3 for S_3 and t_4 for S_4) to change to γ_{new} criticality level.
At date t_1, we are sure that all nodes received the multicast message m. Given that there is a potential clock jitter of ϵ between all nodes, the date t_{change} is not exactly the same in all nodes (compared to the clock reference). It means that there is a time window of size ϵ during which each node is likely to change its criticality level. As a conclusion, we can guarantee that, at date $t_{change} + \epsilon$, all nodes will have increase their criticality level to γ_{new}.

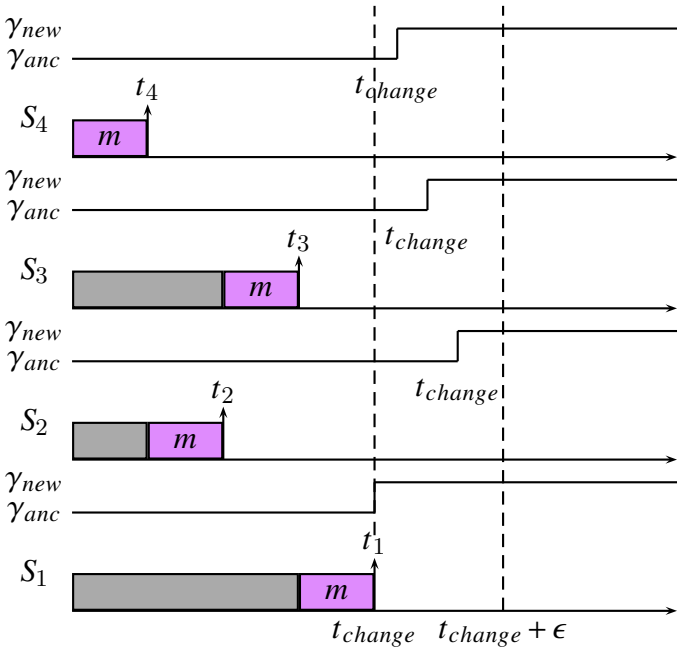


Figure 7.6: Switch criticality level date

This defines a multicast solution assuring that, at a precision based on clock accuracy ϵ, all nodes in

the network change at almost the same instant t_{change}. The computation of this instant is detailed in section 7.2.4.

7.2.3 Decreasing the criticality level

The opposite case to analyze is when the new level γ_{new} is less critical than γ_{anc}. This supposes that the triggering node transmitted a message to the central node which has a lower criticality level. Basically, our protocol assumes that all messages from lower criticality level are dropped out from queues. But, the situation of $\gamma_{new} < \gamma_{anc}$ suggests that the protocol has to take into account the situation when the γ_{anc} critical phase is ended, and we have to decrease the criticality level of the network.
Detecting a message with a higher criticality level is an exception. When this exceptional situation happens, there is no alternative solution: it means that the network has to increase its criticality level. On the contrary, receiving a message with a lower criticality level, or a low transmission time, does not necessarily means that we need to decrease the current criticality level on the network. For example, we can receive a critical message whose transmission delay does not exceed the WCAT corresponding to the current criticality level. We can have short messages corresponding to high criticality levels.
That is why changing back to a lower criticality level implies to determine if there is still higher critical messages in the topology, and if these messages necessarily induce to maintain the criticality level at its highest. We propose two different solutions to manage this information: the full-centralized MC management mode and the half-centralized MC management mode.

Full-centralized management

Full-centralized management proposes to manage MC with an information management entirely done in the central node. It is a solution to integrate criticality level decrease inside a network. This solution is based on the following assumption: all nodes have to keep the central node informed, permanently, of the criticality messages they receive.
The problem is as follows: we want to change the criticality level of all nodes in the network from γ_{and} to γ_{new}, and γ_{new} is considered as a lower criticality level compared to γ_{anc}. That means we must define a process to decide whether or not we can decrease the criticality level of the network to γ_{new} by sending a multicast criticality change message.
The solution proposed in full-centralized management is based on the following assumption: critical messages exceeding their γ_{anc}-WCAT are rare. At each time a node receives a γ_{anc} message, the node transmits a SCC message to the central node. This SCC must contain the value of γ_{anc}. By following this process, the central node will be kept informed permanently if there is still γ_{anc}-critical traffic in the network. As long as the central node gets SCC for γ_{anc} level, it means that the node who transmitted these SCC is calling to stay in γ_{anc} level.
As far as the central node does not receive, from a specific node n, any γ_{anc} message during a certain period of time, it means that there is no more γ_{anc}-critical traffic in n. The criticality level of the network can change back to γ_{new}. The waiting delay without any reception of a γ_{anc} SCC should be representative of all the network flows. For periodic flows, we propose to express this waiting time T_w as the maximum period of each flows in the network. It is represented with the following expression:

$$T_w(\gamma_{new}) = \max_{v_i \in \mathcal{N}}(T_i) \quad (7.5)$$

Once there has been no γ_{anc} SCC message during this delay in the central node, we multicast a criticality level change to γ_{new}. With the reliable multicast protocol, all nodes change their criticality level to γ_{new}. For sporadic flows, we cannot guarantee any maximum delay between two messages emission. It means that the delay $T_w(\gamma_{new})$ is not necessarily sufficient to manage criticality level switches back to lower levels in the case of sporadic flows. Except from an arbitrary solution (for example indexing $T_w(\gamma_{new})$ value on minimum inter-arrival values) we cannot guarantee criticality level changes delay (to lower levels) for sporadic flows.
Nevertheless, this full-centralized solution represents a problem: each time a node receives a γ_{anc}-critical message, the node has to transmit a SCC message to the central node. This represents additional network traffic, particularly when managing a high amount of flows. As a result, centralizing all criticality level change descisions in the central node can strongly impact the network performances. As a conclusion, the full-centralized method can appear to be efficient for dual-criticality levels networks, but its implementation will quickly appear as costful when there is more different levels.

Half-centralized management

The second method proposes to manage decreases in the criticality level by delegating a part of the criticality management to each node. Instead of centralizing all the process to the central node, each node is responsible for deciding to which criticality level it has to change to. Then, the central node is responsible for the final decision (see figure 7.7).
Instead of implementing this waiting delay inside the central node, we store it in each node. The process consists in transmitting a SCC to the central node when a switch did not receive a message of γ_{anc} level during a certain period of time.
This process dedicates a specific waiting delay for each node n in the network $\mathcal{N}$. We do not need anymore to assure that the properties of each flow of the network $\mathcal{N}$ are known by the central node. Each node has to save the WCAT vector and the period (for WCAT-oriented modelling) of each flow that transits through the node. For each node n, its waiting delay can be defined as the maximum period of all flows transiting through n:

$$T_w^n(\gamma) = \max_{v_i, n \in \vec{\mathcal{P}}_i}(T_i^{\gamma}) \quad (7.6)$$

Applying algorithm 1 in the central node, at each SCC message reception, allows us to consider potential changes to lower criticality levels. As a conclusion, the solution proposed by centralized MC management protocol with half-centralization process among the nodes integrates criticality levels management and changes, either to increase or decrease the criticality level of the network. In order to validate the guarantees offered by such a protocol, we now detail the criticality change delay to change the criticality

We keep the same principle as for full-centralized model: we make each node wait during a certain period of time without any γ_{anc}-critical message before triggering any decision. We dedicate a specific waiting delay for each node n.

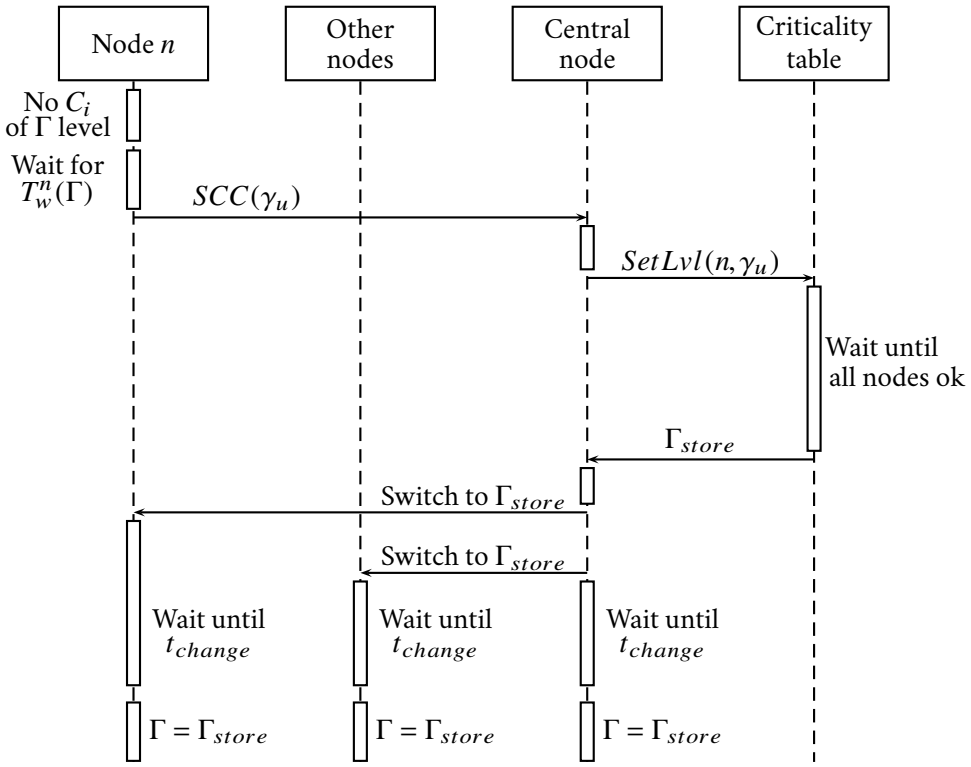


Figure 7.7: Decrease criticality level

level in the network.

7.2.4 Delay computation

In order to provide guarantees to high critical messages transmission with this MC management protocol, transmission delay of all messages has to be bounded. Also, the maximum delay needed to transmit a γ_{new} critical message in the network (in case of an increase in the criticality level) has to be bounded and expressed. In the following part, we want to compute the delay needed to change the criticality level inside the network, from γ_{anc} to γ_{new} level.
As we shown, the centralized MC management protocol works in two phases: call and share. We propose to compute the individual delay of each of these phases. In order to do so, we assume the following statements:

- The SCC is emitted by a node n in the network $\mathcal{N} = \{S_1, S_2, ..., S_k\}$, composed of k switches.
- S_k is designated as the central node in the network.
- We express the delay of the call phase as the delay to detect a WCAT overrun (or shorter period) and to transmit a SCC message from n to S_k. We note this delay as $I(n)$.
- We express the multicast delay (sharing phase) as the delay to multicast the new criticality level γ_{new} to all nodes in the network and to eventually change the criticality level of all the nodes. This delay is the combination of the delay to receive the multicast message in each node. For each node n, the delay required to be ready to change its criticality level is denoted as $M(n)$.

When a node n detects that no message of level γ_{anc} was received during $T_w^n(\gamma_{anc})$, it transmits a SCC message to the central node. The central node receives it and stores the information that node n is currently calling for a criticality level decrease. If node n receives a message of γ_{anc} level, n resets its waiting time.
At each time, the central node has to keep a trace of each criticality level each node calls to change to. This trace log is called the criticality table and is stored in the central node. Each node keeps a trace of each message transiting through it. As long as the criticality level of these messages is lower than γ_{anc}, we store it. When transmitting a SCC message to the central node to call for a decrease in the criticality level, the criticality level information transmitted corresponds to the highest criticality level among all received messages during the delay $T_w^n(\gamma_{anc})$.
The criticality table is a log table, showing the different criticality levels each node called to change to. As soon as all nodes called for a change to a level lower than γ_{anc}, then the central node multicasts a criticality level change to all nodes. The level to change to (γ_{new}) is the highest criticality level stored in the criticality table. When ordering this multicast, the criticality table is reset to γ_{new} for all nodes.
Algorithm 1 manages this criticality switch.

```
Data: Current criticality level γ_anc
Result: Criticality management
wait ← 0
Γ_store ← Γ_LO
while network up do
    m ← pick(queue)
    γ_new ← Criticality(m)
    if γ_new == γ_anc then
        wait ← 0
    else
        if γ_new > γ_anc then
            γ_anc ← γ_new
            Send(SCC, γ_anc)
            Γ_store ← Γ_LO
            wait ← 0
        else
            wait ← wait + 1
            if γ_new > Γ_store then
                Γ_store ← γ_new
            end
            if wait == T_w^n(γ_anc)
             then
                Send(SCC, Γ_store)
            end
        end
    end
end
```

Algorithm 1: Storing lower criticality levels

The global delay to change the criticality level from γ_{anc} to γ_{new}, starting from a SCC emitted by node n, is denoted as $S(n)$. We can express this delay with the following expression:

$$S(n) = I(n) + M(n) \tag{7.7}$$

Our goal is to compute the worst case transmission delays of critical messages in the network. It means that we want to compute the worst case value of $S(n)$, depending on all nodes of the network $\mathcal{N}$. This worst case delay is denoted as $S(\mathcal{N})$ and represented by:

$$S(\mathcal{N}) = \max_{n\in\mathcal{N}}(S(n)) = \max_{n\in\mathcal{N}}(I(n)) + \max_{n\in\mathcal{N}}(M(n)) \tag{7.8}$$

We showed that the protocol does not manage criticality increase and decrease with the same process. The criticality level switching delay will not be the same in these two situations. We first want to char-

acterize the expression of $S(\mathcal{N})$ when increasing the criticality level from γ_{anc} to γ_{new}. Our goal is to assure the consistency of the criticality level information in all nodes of the network.

Increasing criticality level

Call phase delay $I(n)$ includes the delay to detect the criticality of an incoming message, and the delay to send a SCC message to the central node. We consider that the delay to read the 802.1Q tag of a message and detect its criticality level is null.
We also consider that the delay to compute a message size is null. As a conclusion, $I(n)$ can be represented as the worst case end-to-end transmission delay of a SCC message from node n to the central node S_k. The SCC message is characterized as a message c, with the follwing properties: $\{\vec{\mathcal{P}}_c, C_c, T_c\}$. We consider that the value of C_c does not depend on the criticality level.
The SCC message is sent in the VLAN of highest priority. This VLAN is dedicated for configuration purposes and the only messages with a higher priority we can find are PTP messages. It means that the SCC is likely to be delayed by PTP messages. Considering the delay induced by PTP messages in the network, the expression of $I(n)$ can be represented as the latest starting date from central node S_k of the sent SCC message. This delay can be expressed with the Trajectory Approach. We obtain:
The first node of the path $\vec{\mathcal{P}}_c$ is denoted as $first_c$. Given that node transmitted message c, we have $first_c = n$.

$$I(n) = \max_{t \geq 0} \left\{ W_{c,t}^{S_k} - t + C_c \right\}$$

$$W_{c,t}^{S_k} = \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_c \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(1 + \left\lfloor \frac{t + A_{c,j}}{T_j} \right\rfloor \right)^+ * \max(C_j^{\gamma_{new}}, C_j^{\gamma_{anc}}) \quad (7.9)$$

$$+ \sum_{k \in \vec{\mathcal{P}}_c \setminus \{S_k\}} (\max_{k \in \vec{\mathcal{P}}_j} (C_j^{\gamma_{anc}}, C_j^{\gamma_{new}})) \quad (7.10)$$

$$+ \sum_{k \in \vec{\mathcal{P}}_c \setminus \{S_k\}} \mu_k^i \quad (7.11)$$

$$+ (|\vec{\mathcal{P}}_c| - 1) * sl \quad (7.12)$$

$$- \left(\sum_{j \in \vec{\mathcal{P}}_c \setminus \{n\}} (\Delta_{c,t}^j) - t \right)^+ \quad (7.13)$$

$$- C_c \quad (7.14)$$

- Term 7.9 is the delay induced by messages arriving sooner or at the same time compared to message c and which path has at least one common node with path $\vec{\mathcal{P}}_c$. $A_{c,j}$ is equal to the number of messages transmitted from flow v_j likely to delay the transmission of message c.
- Term 7.10 is explained by the required time to transmit message with no additional traffic. This delay is upper-bounded by the size of each already encountered message in a node.

- Term 7.11 is the delay due to non-preemptive effect.
- Term 7.12 is the delay due to switching latency in each link along $\vec{\mathcal{P}}_c$.
- Term 7.13 is the delay due to serialization. The delay due to serialization effect is induced in all nodes of $\vec{\mathcal{P}}_c$, except from the first one denoted as $first_c$. We have $n = first_c$. Term 7.13 is explained by the point that we compute the latest response time, in node S_k, and not the global end-to-end transmission delay. This was detailed in chapter 4.

As mentioned in part II, we consider that messages transmissions in a network are non-preemptive. It means that SCC message transmission delay must take into account the delay due to non-preemptive effect.
We mentioned that SCC message was a message dedicated to MC management in the network. These messages have to be analyzed first and so, attached to the highest possible priority. In order to do so, we transmit the SCC message in a VLAN associated with the highest priority in the network. This is the same VLAN used for PTP messages transmission. As a conclusion, only PTP messages can have the same priority as the SCC message. It means that, as soon as we can compute the number of PTP messages generated during the transmission of a SCC message, we can propose a simplified expression of $I(n)$.
In order to compute the delay induced by PTP messages, we define F_{PTP} the PTP synchronization frequency (statically-defined by the user), and we consider C_{PTP} the WCAT of a PTP synchronization message. In our approach, we consider that all messages related to PTP protocol have the same WCAT.

$$
\begin{aligned}
W_{c,t}^{S_k} = & \sum_{h \in \vec{\mathcal{P}}_c} \left(1 + \left\lfloor \frac{t + A_{c,PTP}}{T_{PTP}} \right\rfloor\right)^+ * C_{PTP} \\
& + \sum_{k \in \vec{\mathcal{P}}_c \setminus \{S_k\}} \max_{k \in \vec{\mathcal{P}}_j} (C_j^{\gamma_{new}}, C_j^{\gamma_{anc}}) \\
& + \sum_{k \in \vec{\mathcal{P}}_c \setminus \{S_k\}} \mu_k^i \\
& + (|\vec{\mathcal{P}}_c| - 1) * sl \\
& - \left(\sum_{j \in \vec{\mathcal{P}}_c \setminus \{S_k\}} (\Delta_{c,t}^j) - t \right)^+ \\
& - C_c
\end{aligned}
\tag{7.15}
$$

We can assume that PTP synchronization messages and SCC criticality management messages are dedicated to the configuration and both based on the same model. This can lead without any major pessimism to assume that both messages have the same size (which is of a few bytes of data plus the Ethernet header). Part II details PTP messages structure. As a conclusion, we assume without loss of generality that $C_c = max(C_c, C_{PTP})$.

Reliable multicast delay As soon as we computed the delay to transmit the SCC message, we now have to characterize the multicast delay $M(n)$. This delay computes the delay needed between the reception of the SCC message and the date when a node n is ready to change its internal criticality level. Given that the criticality level information is multicasted and all nodes change their criticality level at a date t_{change} (with an accuracy of clock ϵ), it means that the effective multicast delay of the protocol is equal to the worst case of $M(n)$ among all nodes n. We consider that the delay to update the criticality level value inside a node n is null.
We note the global multicast delay as $M(\mathcal{N})$, and its expression is as follows:

$$M(\mathcal{N}) = \max_{n \in \mathcal{N}}(M(n)) \tag{7.16}$$

For each node n, we need to compute the delay to transmit the multicast message from the central node to n. This transmission, as it goes in the opposite way as network traffic, is not delayed by network traffic (assuming that all network links are full duplex). We compute $M(n)$ as the addition of the following elements:

- The electronical switching latency sl induced by the electronical transmission between two nodes.

- The WCAT of the multicast message, denoted C_o.

For a node n in the network $\mathcal{N}$ with a central node S_k, the delay needed to receive the multicast order directly depends on the distance between n and the central node S_k. We note this distance as $d_n^{S_k}$. We assume in this work that sl is a worst case evaluation of each switching latency, for each switch, and is considered as a constant.
It takes a certain delay to transmit the multicast message m to a node n in the network. The reliable multicast delay in the network is the worst case of all the delays, for all nodes n in the network $\mathcal{N}$. We use the Trajectory Approach to express the multicast transmission delay $M(n)$ for each node n.
In the general case, we obtain the following expression:

$$M(n) = \max_{t \geq 0} \left\{ W_{m,t}^{S_k} - t + C_c \right\}$$

$$W_{m,t}^{S_k} = \sum_{\substack{j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}_m} \cap \vec{\mathcal{P}_m} \neq \emptyset}} N_{m,j}^{first_{m,j}} * C_c \quad (7.17)$$

$$+ \sum_{k \in \vec{\mathcal{P}_m} \setminus n} \max_{k \in \vec{\mathcal{P}_j}} (C_j^{\gamma_{new}}, C_j^{\gamma_{anc}}) \quad (7.18)$$

$$+ \sum_{k \in \vec{\mathcal{P}_c} \setminus \{S_k\}} \mu_k^i \quad (7.19)$$

$$- \left(\sum_{j \in \vec{\mathcal{P}_m} \setminus \{S_k\}} (\Delta_{m,t}^j) - t \right)^+ \quad (7.20)$$

$$+ (d_n^{S_k} - 1) * sl - C_c \quad (7.21)$$

- The term 7.17 is the delay represented by messages arriving sooner or at the same time compared to the multicast message.

- The term 7.18 is explained by the required time to transmit message with no additional traffic. This delay is upper-bounded by the size of each already encountered message in a node.

- The term 7.19 represents the potential non-preemptive effect induced by other messages in the network.

- The term 7.20 is the serialization effect of message. Finally, the term 7.19 is the delay of transmission of the multicast through all the nodes from S_k to n, including the switching latency sl.

Now, we consider that the multicast message m is sent in the VLAN with the highest connection. It means that only PTP messages are likely to be with a higher priority as the message m (like it was the case for SCC). m is characterized by $\{\vec{\mathcal{P}_m}, C_m, T_m\}$. For readability purposes, we consider that SCC c and multicast messages m have the same WCAT: $C_c = C_m$.

$$W_{m,t}^{S_k} = \sum_{k \in \vec{\mathcal{P}_m}} N_{m,PTP}^{first_{m,PTP}} * C_c \tag{7.22}$$

$$+ \sum_{k \in \vec{\mathcal{P}_m} \setminus n} \max_{k \in \vec{\mathcal{P}_j}} (C_j^{\gamma_{new}}, C_j^{\gamma_{anc}}) \tag{7.23}$$

$$+ \sum_{k \in \vec{\mathcal{P}_m} \setminus \{n\}} \mu_k^i \tag{7.24}$$

$$- \left(\sum_{j \in \vec{\mathcal{P}_c} \setminus \{S_k\}} (\Delta_{m,t}^j) - t \right)^+ \tag{7.25}$$

$$+ (d_n^{S_k} - 1) * sl - C_c \tag{7.26}$$

The term 7.22 is the delay induced by PTP messages, having a higher or equivalent priority compared to message m. We have $N_{m,PTP}^{first_{m,PTP}} = \left(1 + \left\lfloor \frac{t + A_{m,PTP}}{T_{PTP}} \right\rfloor\right)^+$. This term is the number of PTP messages likely to delay message m in each node.

As we are computing the multicast delay, we need to compute the path to each node required to multicast the criticality information. Moreover, we have to know which switch is the farthest from the central node, to know which node will get the multicast MC information at last. At the end of this multicast delay $M(\mathcal{N})$, we are sure that all the nodes in the network received the criticality change information. The criticality level change happens on any node at its local date $t_m + M(\mathcal{N}) + \epsilon$ where t_m is the timestamp sent by the central node in the criticality change multicast message.

To preserve the consistency of the network, we have to wait for all nodes to receive this multicast before ordering a criticality level switch. At the date $t_m + M(\mathcal{N}) + \epsilon$, we can assume that all the nodes switched their criticality level. Given this and the expressions of $I(n)$ and $M(n)$, we can deduce the expression of $S(\mathcal{N})$. We consider that C_c and sl are constant in the network. We obtain the following expression:

$$
\begin{aligned}
S(\mathcal{N}) &= \max_{n\in\mathcal{N}}(I(n)) + \max_{n\in\mathcal{N}}(M(n)) \\
&= \max_{\substack{n\in\mathcal{N}\\ t\geq 0}}(W_{c,t}^{S_k} - t + C_c) + \max_{\substack{n\in\mathcal{N}\\ t\geq 0}}(W_{m,t}^{S_k} - t + C_c) \\
&= \max_{\substack{n\in\mathcal{N}\\ t\geq 0}}\Bigg[\sum_{h\in\vec{\mathcal{P}_c}}\left(N_{n,PTP}^{first_{n,PTP}} * C_c\right) + \sum_{k\in\vec{\mathcal{P}_c}\setminus\{S_k\}}\max_{k\in\vec{\mathcal{P}_j}}(C_j^{\gamma_{new}}, C_j^{\gamma_{anc}}) \\
&+ \sum_{k\in\vec{\mathcal{P}_c}\setminus\{S_k\}}\mu_k^i - \left(\sum_{j\in\vec{\mathcal{P}_c}\setminus\{n\}}(\Delta_{c,t}^j) - t\right)^+ + (|\vec{\mathcal{P}_c}| - 1) * sl - t\Bigg] \\
&+ \max_{\substack{n\in\mathcal{N}\\ t\geq 0}}\Bigg[\sum_{k\in\vec{\mathcal{P}_m}} N_{n,PTP}^{first_{n,PTP}} * C_c + \sum_{k\in\vec{\mathcal{P}_m}\setminus\{n\}}\max_{k\in\vec{\mathcal{P}_j}}(C_j^{\gamma_{new}}, C_j^{\gamma_{anc}}) \\
&+ \sum_{k\in\vec{\mathcal{P}_m}\setminus\{n\}}\mu_k^i - \left(\sum_{j\in\vec{\mathcal{P}_m}\setminus\{S_k\}}(\Delta_{n,t}^j) - t\right)^+ + (d_n^{S_k} - 1) * sl - t\Bigg]
\end{aligned}
\tag{7.27}
$$

We suppose that $\forall n \in \mathcal{N}, C_o^n = C_o = C_c$. Based on the expression of $S(\mathcal{N})$, we can assume for all nodes in the network that, starting from an overrun detection at a date t, we can assure that all nodes in the network $\mathcal{N}$ will have switch their criticality level from γ_{anc} to γ_{new} at a date $t + S(\mathcal{N}) + \epsilon$.
As a conclusion, we show that the delay needed to change the criticality level from a γ_{anc} to γ_{new} when $\gamma_{new} > \gamma_{anc}$ is upper-bounded by $S(\mathcal{N})$. This assures the reliability of our multicast protocol. Depending on the value of γ_{anc} and γ_{new}, the value of $I(n)$ is likely to change. We can bound criticality level changes delay in all multi-criticality level networks with the proposed protocol. Centralized MC management protocol is compliant to our requirements in terms of MC management.

Decreasing criticality level

In order to define the delay to decrease the criticality level inside the network $\mathcal{N}$, we need to base our approach on the following question: starting from the moment when there is no more critical messages in the network (no critical message currently transmitted and no critical message waiting to be transmitted in any switch queue), how long does it take to change back to a lower criticality level in all nodes?
The two approaches we defined previously (full-centralized and half-centralized, see section 7.2.3) propose different solutions to manage this criticality level change from γ_{anc} to γ_{new}. We detail the delay required by each approach to operate this switch.

Exemple We want to compare both approaches on a simple example, in order to illustrate the differences. We consider the topology represented in figure 7.8 which flows parameters are detailed below. We consider a dual-criticality level network (LO, HI), and we set the WCAT of a SCC is equal to $10\mu s$ (C_c). In this example, we suppose that clock synchronization as an accuracy of $\epsilon = 0$ for readability purposes. Given the detailed parameters, we obtain the results described in figure 7.9. At each reception of a HI-critical message in a node, the waiting delay is reset to 0. In the central node S_4, we change the current state of each node (S_1, S_2, S_3, S_4) in the criticality table: each time we receive a SCC from one node, the

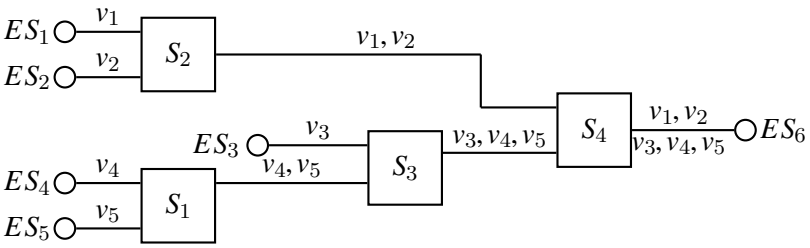


Figure 7.8: Example topology

v_i	$C_i^{LO}(\mu s)$	$C_i^{HI}(\mu s)$	$T_i(\mu s)$
v_1	20	30	100
v_2	20	-	90
v_3	20	-	100
v_4	20	-	50
v_5	20	30	80

state of this node in the criticality table is updated to the criticality level this node calls to change to. The value of this level is contained in the SCC. Once all nodes sent a SCC to ask for a change back to LO, the central node triggers a multicast to LO-criticality mode.

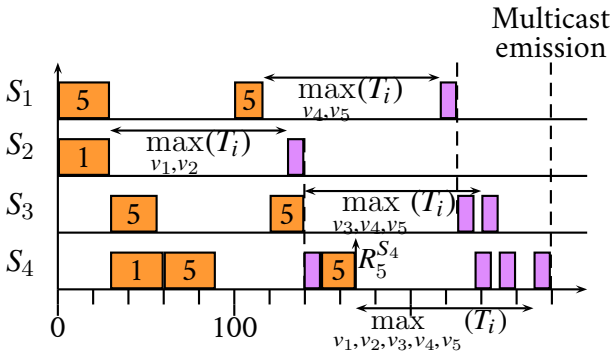


Figure 7.9: Switch criticality from HI to LO

We can see in figure 7.9 that a message which path is shared among different nodes in the network can have its period taken into account several times in the same computation of the waiting delay. For example, the message 5 is taken into account for the waiting delay of the nodes S_1, S_2 and S_4.
In the example, the delay to change back to LO mode is the instant when the last node (S_4) finished to receive and transmit the SCC message. This date is computed based on the emission of the latest SCC message to LO mode before effective multicast emission.

In the example, the delay between the latest SCC message emission and the effective multicast emission is computed as the reception instant of the latest SCC message in S_4 minus $R_5^{S_4}$, which gives us: $300-170=130\mu$s. If we apply the multicast computation delay given previously, we obtain a global changing delay equal to: $130+(C_o+sl)*\max_{n\in\mathcal{N}}(d_i^n)=130+2*10=150\mu$s.
We can assume this delay as the delay during which the network was in HI mode without HI messages traffic transmission. The longest of these delays among all nodes of the network will give us the total delay lost in waiting to change back to LO mode.

Delay computation Based on this example, we compute the delay needed to change to a lower criticality level, starting from the instant at which there is no more critical messages transmissions in the network. The value of the current level γ_{anc} has an impact on this delay.
The criticality management protocol we defined allows us to manage criticality level switches even for decreasing the current criticality level. We need to assure that there is no more critical messages transmission before decreasing the current criticality level, inducing a waiting delay of $\max_{v_i\in\mathcal{N}}(T_i^{\gamma_{anc}})$. This waiting delay has to be taken into account in the criticality level changing delay.
We observe that the multicast delay is exactly the same as when we want to increase the criticality level. If we focus on a full centralized protocol, we can note the delay $S^{\gamma_{anc}}(\mathcal{N})$ as the delay needed to change from γ_{anc} criticality level to a lower level. This delay is computed as follows:

$$S^{\gamma_{anc}}(\mathcal{N}) = \max_{v_i \in \mathcal{N}}(T_i^{\gamma_{anc}}) + S(\mathcal{N}) \quad (7.28)$$

7.3 Transition phase

We want to on how to manage critical messages transmission during the change of criticality level inside the nework. As shown in section 7.2.4, changing the criticality level inside a network with this centralized MC management protocol requires a delay which had been specified in both situations: when we have to increase the criticality level, and when we have to decrease it.
When increasing the criticality level, the delay required is the delay to call for a criticality level change and the time to multicast the new criticality level information. The order to change is directly taken into account and the multicasted message to increase the criticality level is immediately transmitted (see section 7.2.4). Nonetheless, the delay to change the criticality level induces a transition phase for the network, during which there are mixed receptions of critical and non critical messages inside nodes. In order to assure the transmission of critical messages during this phase, we want to specify the scheduling policy in the nodes during the transition.
The transition phase starts when the criticality level is at γ_{anc} and a node detects a message m_i from a flow v_i which exceeds its γ_{anc} WCAT. We suppose that we are in the case of a need to increase the criticality level of the network $\mathcal{N}$, so

$$\forall v_i \in \mathcal{N}, C_i^{\gamma_{anc}} \leq C_i^{\gamma_{new}} \quad (7.29)$$

This WCAT overrun (or shorter period) detection happens in a node $S_j \in \mathcal{N}$. In all other nodes in the network, as long as there is no other γ_{new} message detection, the detection of this overrun has no impact on their traffic management or scheduling policy. If there is a need to transmit a SCC from another node which also detected a WCAT overrun, then this node will also be concerned by a potential mix between flows of different criticality during the induced transition phase. In the following part, we call all the nodes that are detecting a need to change the criticality level the detecting nodes.
During this transition phase, the nodes in the network are not yet informed that the criticality level of the network is about to change. It means that we cannot expect from these nodes to modify their behabior. As a result, critical messages are likely to be delayed by non critical traffic issued from other nodes during this phase. That leads to the following question: how do we assure the transmission of γ_{new} messages during criticality transition phase? We propose different approaches to answer to this question.
The first approach is the non-blocking approach. It consists in transmitting γ_{new} critical messages as soon as they are received, even during the transition phase. During the transition phase, these γ_{new} critical messages are considered part of the traffic as all other messages. They will be seen as WCAT exceeding, triggering SCC emissions in all nodes along their path. There is no specific reason for these γ_{new}-critical messages to be transmitted with the highest priority in the network. We have to define the transmission

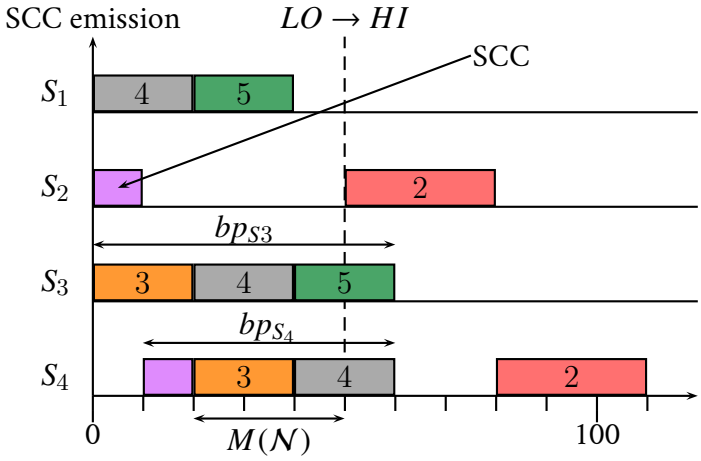
delay induced to γ_{new}-critical messages during this phase.
The second approach is the blocking approach. It proposes to entirely stop the transmission of γ_{new}-critical messages from the detecting nodes, and to wait for the end of the transition phase to send the γ_{new}-critical messages. With this solution, we can assure that γ_{new}-critical messages will not be delayed by γ_{anc}-critical messages still present in the topology. This solution is called the blocking approach.
We expose and detail here both approaches and to compare their impact on γ_{new} critical messages scheduling. Particularly, we study what approach provides the better worst case end-to-end transmission delay for γ_{new}-critical messages. In the section below, we detail the two approaches issued from the application of the centralized protocol and we compute worst case end-to-end transmission delay resulting from both approaches.

7.3.1 Blocking approach

The blocking approach is the first method. As long as the criticality level of the network is not switched to γ_{new} in every node of the network, all messages (γ_{new}-critical and γ_{anc}-critical) are blocked in the detecting node.
It means that there is a traffic interruption inside each detecting node, changing them to an idle mode, waiting for the criticality level of the network to change to γ_{new}.
γ_{new}-critical messages transmission in the blocking approach do not suffer from additional delay induced by γ_{anc}-critical level. We add the delay represented by criticality level change to the γ_{new}-message transmission: this additional delay is represented by the length of the transition phase. Thus, the transmission of γ_{new}-critical messgaes is still delayed by the non-preemptive effect (see section 7.2) induced by messages from all other nodes.
The purpose of this approach is to be able to bound the transmission delay of γ_{new}-critical messages during criticality level switches (the transition phrase). The blocking approach assures that any γ_{new}-critical message will not be delayed more than the duration of the transition phase.

Example

We focus on the results shown in figure 7.10.



With this approach, we obtain a transmission delay of $R_2^{S_4} = 110\mu$s, representing 20μs more than for the non-blocking approach. We observe in the node S_2 that, as soon as WCAT exceeding for message 2, the node S_2 stops emitting messages (exception for SCC, dedicated for criticality configuration). Message 2 is not delayed by messages with a higher or equivalent priority.

Figure 7.10: Blocking approach example

Delay computation

We want to characterize the worst case end-to-end transmission delay of γ_{new} critical messages during criticality level change transition phase, managed by the blocking approach. The expression of this delay is based on two different terms: first, the delay to change the criticality level in the network, according to the centralized protocol. This delay has been expressed as $S(\mathcal{N})$ in previous section 7.2.4.
Second, we need to add the transmission delay of the critical message itself to the delay. Based on the Trajectory Approach, we can express this delay. It is expressed depending on the γ_{new}-critical traffic in the network, expressed with:

$$\sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(N_{i,j}^{first_{i,j}} * C_j^{\gamma_{new}} \right) \tag{7.30}$$

Based on this expression, we obtain the worst case end-to-end transmission delay of γ_{new}-critical message. This delay considers that all γ_{anc}-critical messages can induce a non-preemptive effect which will be added to the global expression. The delay to transmit a γ_{new}-critical message with the blocking approach is denoted as $R_i^B(\gamma_{anc}, \gamma_{new})$. It can be expressed as follows:

$$\begin{aligned}
R_i^B(\gamma_{anc}, \gamma_{new}) &= \max_{t \geq 0} \{ W_{i,t}^{last_i} - t + C_i^{\gamma_{new}} \} \\
W_{i,t} &= S(\mathcal{N}) \\
&+ \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(N_{i,j}^{first_{i,j}} * C_j^{\gamma_{new}} \right) \\
&+ \sum_{k \in \vec{\mathcal{P}}_i} \left(\max_{k \in \vec{\mathcal{P}}_j} (C_j^{\gamma_{anc}}, C_j^{\gamma_{new}}) \right) \\
&+ \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu_k^i \\
&- \left(\sum_{j \in \vec{\mathcal{P}}_i \setminus first_i} (\Delta_{i,t}^j) - t \right)^+ \\
&+ (|\vec{\mathcal{P}}_i| - 1) * sl \\
&- C_i^{\gamma_{new}}
\end{aligned} \tag{7.31}$$

One first assumption would be to consider this approach as pessimistic compared to the non-blocking approach.We add to the transmission of each γ_{new}-critical message the delay due to the transition phase length. This additional delay can be considered as a source of pessimism.
The blocking approach assures the transmission of γ_{new}-critical messages by assuring that their transmission will not be delayed by γ_{anc}-critical traffic. It means that the γ_{new}-critical messages are released

later, but will suffer for a shorter delay from network traffic (only due to a potential non-preemptive effect).
If we focus on the above example, and change $C_1^{LO} = 50\mu$s. We then obtain $R_2^B(t) = 50 + (20 + 20) + 20 = 110\mu$s for the blocking approach (the delay does not change, as message 2 is not delayed by message 1), and $R_2^{NB}(t) = 120\mu$s for the non-blocking approach. What we observe is that, depending on the traffic configuration in the detecting node, the approaches can provide a shorter delay compared to the other, or on the opposite appear to be way longer. In order to compare the transmission delay of both approaches, we expose below criterias of comparison between them.

7.3.2 Non-blocking approach

The non-blocking approach is the second proposed method to schedule γ_{new}-critical messages transmission during the criticality change transition phase from γ_{anc} to γ_{new}. The non-blocking approach consists in transmitting γ_{new}-critical message like any other message in the network. This creates a mixed transmission of γ_{new} and γ_{anc}-critical messages, until the effective criticality level change to γ_{new} level becomes effective.
With the non-blocking approach, we do not have to take into consideration the criticality level change delay as an additional delay in the transmission of γ_{new}-critical messages. As these messages are transmitted during the transition, there is no waiting delay added to the transmission of γ_{new}-critical messages. There is no addition of the criticality change delay to γ_{new} critical messages worst case end-to-end transmission time.
On the opposite, γ_{anc}-critical messages in the network represent an additional delay which has to be taken into account when computing γ_{new}-critical messages end-to-end transmission delays.

Example

As an introduction, we want to illustrate the non-blocking approach principle on an example. We consider the network $\mathcal{N}$ presented in section 7.8. In this example, we suppose that there are two possible criticality levels in the network, denoted LO and HI. The details of the different flows in network $\mathcal{N}$ are indicated below.
We suppose that C_c, the common WCAT of the criticality level change multicast order and SCC message, is equal to 10 μs. We consider both these WCAT as equal for readability purposes (see section 7.2.2 for more details on this point).

v_i	C_i^{LO} (μs)	C_i^{HI} (μs)	T_i (μs)
v_1	20	-	200
v_2	20	30	200
v_3	20	-	200
v_4	20	-	200
v_5	20	-	200

We suppose that $sl = 0\mu$s. We consider PTP traffic as negligible for readability purposes, and we finally assume that all clocks are perfectly synchronized ($\epsilon = 0$). We obtain the results detailed in figure 7.11.

The computation of the global criticality level change delay from LO to HI is equal to $S(\mathcal{N}) = I(S_4) + M(\mathcal{N})) = 20 + 40 = 60\mu$s, with a multicast delay $M(\mathcal{N}) = d_{S_1} * C_c = 3 * 10 = 30\mu$s. In this example, we observe that the transmission of 2 is delayed by LO-critical traffic issued from flow v_1 in node S_2

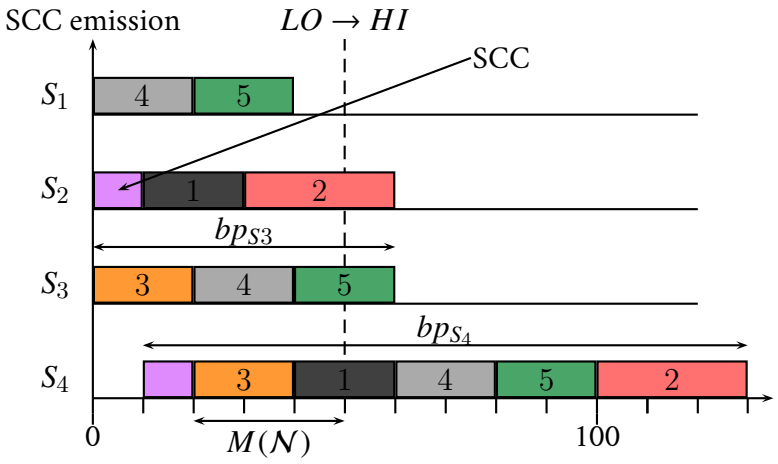


Figure 7.11: Non-blocking approach example

The message 2 is a HI-critical message from flow v_2. We compute its worst case end-to-end delay between its source node ES_2 and S_4 (the last node in its path). We obtain: $R_2 = 90\mu s$. In this end-to-end transmission time, the delay induced by LO-critical traffic is equal to: $C_1^{LO} + C_c * C_3^{LO} = 50\mu s$, integrating the delay due to SCC message transmission ($C_c = 10\mu s$).
During the transmission of messages in S_4, criticality level change happens at $t = 50\mu s$.

(see figure 7.11). Based on this example, we conclude that the delay experienced by HI-critical messages during the transition phase depends on the LO-critical traffic. We want to compute the maximum delay represented by the non critical traffic.

Delay computation

During the criticality level transition phase with the non-blocking approach, the transmission of a γ_{new}-critical message has to be considered as the transmission of any other message in the network. It means that despite their higher criticality level, γ_{new} messages have to be considered as potentially delayed by any message with a higher or equivalent priority. As long as the criticality level of the topology has not switched from γ_{anc} to γ_{new}, a γ_{new}-critical message cannot be specifically transmitted with the highest priority.
Worst case end-to-end transmission delay computation of a γ_{new}-critical message has to consider, basically, the lowest possible priority for the γ_{new} message to send. In order to compute to compute this transition delay, we use the Trajectory Approach. We suppose the network $\mathcal{N}$ composed of a set of flows $v_1, v_2, ..., v_n$. We suppose that we detect a γ_{anc}-WCAT exceeding at time t in the network, triggering a SCC to γ_{new} level.
We note $R_i^{NB}(\gamma_{anc}, \gamma_{new})$ the worst case transmission delay of a γ_{new}-critical message from a flow v_i during criticality level transition phase, observed with the non-blocking approach. We have to consider, in our approach, that every potential γ_{new} message will be transmitted with its γ_{new}-WCAT. The worst case end-to-end delay of a γ_{new}-critical message will be expressed as:

$$R_i^{NB}(\gamma_{anc}, \gamma_{new}) = \max_{t \geq 0}\{W_{i,t}^{last_i} - t + C_i^{\gamma_{new}}\} \quad (7.32)$$

As shown in the above example, we have to take into consideration that the maximum delay induced by messages with a higher or equivalent priority is bounded by the duration of the transition phase. At the end of the transition phase, we stop transmitting γ_{anc}-critical messages, and only send γ_{new}-critical

messages. We use the expression of $S(\mathcal{N})$ given in 7.2.4 and obtain the following expression:

$$
\begin{aligned}
W_i^{last_i} = & \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(N_{i,j}^{first_{i,j}} * \max(C_j^{\gamma_{anc}}, C_j^{\gamma_{new}}) \right) \\
& + \sum_{k \in \vec{\mathcal{P}}_i} (\max_{k \in \vec{\mathcal{P}}_j} (C_j^{\gamma_{anc}}, C_j^{\gamma_{new}})) \\
& + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu_k^i \\
& + (|\vec{\mathcal{P}}_i| - 1) * sl \\
& - \left(\sum_{j \in \vec{\mathcal{P}}_i \setminus first_i} (\Delta_{i,t}^j) - t \right)^+ \\
& - C_i^{\gamma_{new}}
\end{aligned}
\tag{7.33}
$$

The final expression we obtain is the worst case end-to-end transmission delay of a γ_{new}-critical message provided by the Trajectory Approach. This delay supposes that each message in the detecting node potentially delays messages from v_i, either because of higher priority transmission or non-preemptive effect due to messages with a lower priority. This expression is based on the evaluation of all potential γ_{anc}-critical messages likely to delay the transmission of γ_{new}-critical messages.
If we apply this expression on the example given above, we obtain: $R_i(\gamma_{anc}, \gamma_{new}) = (20 + 30 + 20 + 20 + 20 + 10) + (30 + 20) = 160\mu$s. The provided value is an upper bound of the worst case end-to-end transmission delay of 2 with the non-blocking approach. It is based on the hypothesis that message 2 suffers from the maximum γ_{anc} traffic plus the maximum potential non-preemptive effect.

7.3.3 Comparing the approches

We want to compare the transmission delay of a γ_{new}-critical message obtained with both blocking and non-blocking approaches. The comparison of these approaches will allow us to decide which approach to apply on the network traffic during the criticality level transition phase.
We suppose the transmission of a γ_{new} message from a flow v_i, characterized by $\{\vec{\mathcal{P}}_i, \vec{C}_i, T_i\}$ (WCAT-oriented modelling). We suppose that the WCAT exceeding of the message is detected in a node n from network $\mathcal{N}$. By combining the results obtained in 7.28 and 7.30, we want to find situations verifying:

$$
R_i^{NB}(\gamma_{anc}, \gamma_{new}) > R_i^{B}(\gamma_{anc}, \gamma_{new}) \tag{7.34}
$$

As soon as the global network traffic provided by γ_{anc}-critical messages (with a higher or same priority as messages from v_i) provides a loger delay than the global criticality level changing delay $S(\mathcal{N})$, then the non-blocking approach provides a longer transmission delay for critical messages.

We consider all nodes as potential detecting nodes. It means that a γ_{anc}-WCAT exceeding can be detected in several nodes at once during the transition phase. We combine the expressions of $R_i^{NB}(\gamma_{anc}, \gamma_{new})$ and $R_i^B(\gamma_{anc})$. We make two assumptions: the potential non-preemptive effect in both approaches is the same, and the delay induced by the serialization effect is the same too. We obtain:

$$\sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} (N_{i,j}^{first_{i,j}} * C_j^{\gamma_{anc}}) > S(\mathcal{N}) \qquad 7.35$$

This condition is verified when delay due to the traffic with higher or same priority compared to γ_{new}-critical message i is superior than the global duration of the transition phase. In order to extract comparison results by simulation from this condition, we need to compute the value of $S(\mathcal{N})$. $S(\mathcal{N})$ depends on the network size, switch configuration and on the WCAT of the SCC message.
We observe that the value of the WCAT of the SCC message has a strong impact on the decision of which approach to select. It means that the size of the SCC message has to be detailed and precisely defined. The details of the SCC frame are given in chapter 14.1.

Correction of the non-blocking approach

The computation of the non-blocking approach delay is based on the characterization of the γ_{anc} messages during the transition phase. This transition phase starts when we detect a γ_{new}-critical message and ends when all switches did change their criticality level to γ_{new}. If we strictly follow this approach, comparing blocking and non-blocking approaches is synthesized to compare the duration of the transition phase, represented by $S(\mathcal{N})$, and the delay induced by other messages in the network, represented by $\sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(N_{i,j}^{first_{i,j}} * \max(C_j^{\gamma_{anc}}, C_j^{\gamma_{new}}) \right)$.
But the expression of the transmission delay for the non-blocking approach we gave previously (see 7.25 and 7.26) is pessimistic. The computation of this delay is based on the assumption that, for the non-blocking approach, all the γ_{anc}-critical messages received in nodes during the transition phase are likely to be transmitted, even if the starting time of the individual transmission of each γ_{anc}-critical message starts after the end of the transition phase. This is not correct. In real cases, as soon as the transition phase is ended and the criticality level switched to γ_{new}, we stop transmitting γ_{anc} messages. We propose to compute the delay $R_i^{NB}(\gamma_{anc}, \gamma_{new})$ while taking into account this phenomenon.
If we focus on the example in section 7.3.2 we obtain the results of figure 7.12.
The delay computation for γ_{new}-critical messages transmission with the non-blocking approach has to be corrected to take into account this situation and correct a potential source of pessimism. This pessimism comes from the point that, in some case, the expression of the delay considers a too high quantity of γ_{anc} traffic. This traffic is bounded by the duration of the transition phase. We obtain the following expression for $R_i^{NB}(\gamma_{anc}, \gamma_{new})$:

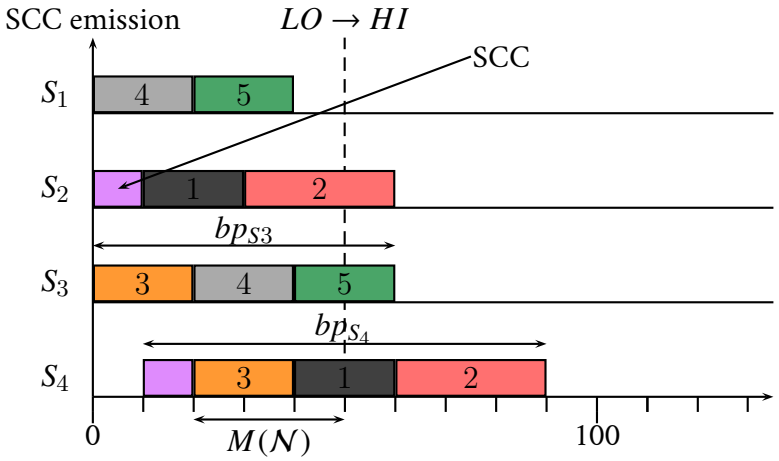


Figure 7.12: Non-blocking approach correction

In that case, messages 4 and 5 are dropped out from S_4 because their release time is after the criticality switch to HI level. It represents an induced pessimism of $C_4^{LO} + C_5^{LO} = 40\mu$s.
As a result, the transmission delay of message 2 is reduced to 90μs. This is lower than the transmission delay provided with the blocking approach (20μs lower).

$$W_{i,t}^{last_i} = \min\Bigg(S(\mathcal{N}), \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} (N_{i,j}^{first_{i,j}} * C_j^{\gamma_{anc}}) \tag{7.36}$$

$$+ \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} (N_{i,j}^{first_{i,j}} * C_j^{\gamma_{new}}) \tag{7.37}$$

$$+ \sum_{j \in \vec{\mathcal{P}}_i} (\max_{k \in \mathcal{P}_j}(C_j^{\gamma_{anc}}, C_j^{\gamma_{new}}))$$

$$+ \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} \mu_k^i$$

$$+ (|\vec{\mathcal{P}}_i| - 1) * sl \tag{7.38}$$

$$- \left(\sum_{j \in \vec{\mathcal{P}}_i} (\Delta_{i,t}^j) - t \right)^+ - C_i^{\gamma_{new}} \Bigg) \tag{7.39}$$

- 7.32 is the delay induced by the messages with a higher or same priority as messages from v_i. This delay is bounded, at the maximum, by the duration of the transition phase, computed with $S(\mathcal{N})$.
- 7.33 is the delay due to non-preemptive effect induced by γ_{anc} and γ_{new}-critical messages.
- 7.34, 7.35 are delays due to serialization effect and switching latency, already detailed in 7.2.4.

This expression, correcting the initial pessimism of the non-blocking approach, assures that the worst case end-to-end delay for a γ_{new}-critical message during the transition phase is necessarily lower or equal to the delay provided with the blocking approach. If we ignore the induced delay due to γ_{new}-critical messages and potential non-preemptive effect (we consider that it is the same with both approaches), we have to compare $\min\left(S(\mathcal{N}), \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(N_{i,j}^{first_{i,j}} * C_j^{\gamma_{anc}}\right)\right)$ and $S(\mathcal{N})$. Obviously, we have the following

property:

$$\min\left(S(\mathcal{N}), \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} \left(N_{i,j}^{first_{i,j}} * C_j^{\gamma_{anc}}\right)\right) \leq S(\mathcal{N}) \tag{7.40}$$

7.4 Conclusion

In this chapter, we presented a centralized protocol as a solution for MC integration inside RT networks. This protocol is based on the centralization of criticality level management in a specific node in the network, considered as a central node. This protocol is based on clock-synchronized architectures, assuring the reliability and the consistency of criticality levels updates in the network.
Through a model of delay computation provided by an application of the Trajectory Approach, we computed an upper bound on the worst case delay needed to change the criticality level in the network. This delay depends on several parameters, such as the network traffic (which can induce a non-preemptive effect) and the switching latency. Given that this delay cannot be considered as null, we had to introduce two approaches, blocking and non-blocking, to propose a solution to guarantee critical messages transmission the criticality level transition phase.
Both these approaches rely on a different policy: either we stop the transmission of critical messages during the transition (blocking approach) or we do not stop them (non-blocking approach). We provided experimental results on the transmission delay computation for both these approaches. These results can be found in chapter 12. These results show that, for high loads, the blocking approach provides shorter transmission delays, especially for critical messages. But, a correction provided for the non-blocking approach tends to show that, in real cases, the blocking approach provides longer delays, due to the requirement to take into account the criticality level change delay in the computation.

Chapter 8

Decentralized MC management

"Nous vénérons le chaos car nous aimons produire de l'ordre."

"We adore chaos because we love to produce order."

– *Maurits Cornelis Escher*

Contents

8.1 Introduction

The centralized protocol presents different problems on which we now focus:

- First, the reliable RT multicast protocol implies a clock-synchronized network. As a result, it implies using a clock-synchronization protocol with a good accuracy such as PTP. Clock-synchronization management inside physical devices is not by itself a common functionality. It means that, in order to integrate the centralized protocol, we should use clock-synchronization compliant network devices, which has an impact on the financial cost of the infrastructures.
- MC integration implies satisfying and guaranteeing weak temporal isolation (see section 6.3.3) and timeliness constraints in the network. As a result, the centralized protocol defined for MC integration in RT networks is based on the process of ignoring all non critical traffic during critical phases. This necessarily represents a loss of data, which drastically decreases the QoS (due to non transmitted messages) of non critical messages in order to assure the respect of the weak temporal isolation constraints of critical flows
- During a criticality level switch, the centralized protocol implies managing a transition phase during which both critical and non critical messages can be simultaneously transmitted. In order to guarantee the weak temporal isolation and the bounded transmission delays for critical messages, we defined specific scheduling approaches for this transition phase.
- Eventually, in order to send and share criticality management informations, the centralized protocol implies generating additional traffic in the network (SCC messages, reliable multicast transmission). This network traffic is dedicated to MC management. It represents an additional increase of the network traffic for configuration purposes and this traffic can lead to an additional decrease of the QoS by delaying messages.

As a conclusion, the centralized protocol has been proved reliable, and it proposes a MC management solution for RT and embeded network architectures. But it represents several problems of costs and QoS losses.
But, it appears to be pessimistic in terms of QoS and ressources costs. As a result, we propose in the following work to present an alternative to this centralized protocol. This alternative protocol is called the decentralized protocol.

8.2 QoS solutions

The main purpose of MC management and integration inside RT networks is to mix critical and non critical functions inside the same infrastructure while guaranteeing the weak temporal isolation and reliability. This can concern dual-critical networks with two different criticality levels (LO, HI), or networks integrating various criticality levels (at least 3) inside the same infrastructure.
Nowadays, one emerging topic [130] consists in focusing on the feasibility of mixing HI-critical reliability constraints and QoS constraints represented by LO-critical traffic, especially during HI-critical phases. In order to propose an answer to this, we want to introduce a new MC management protocol.

The design of the centralized protocol comes from the processor-oriented modelling, where the memory management is centralized. In this context, all cores from a common platform rely on the same criticality information. But it is not necessary to change all nodes to critical mode each time a critical message has to be transmitted. We can provide a solution which would assure critical messages transmissions whereas only changing to critical mode specific nodes.

8.2.1 Example

The purpose of this example is to focus on the impact of centralized MC management on LO-critical messages. We build a network topology as described in figure 8.1. This is a tree-oriented topology organized around a central switch S_4. We suppose this network composed of two criticality levels LO and HI. For all LO-critical flows v_i, we assume that $C_i^{HI} = -1$.
All flows are modeled with WCAT-oriented method: we consider that, no matter the criticality level of the network, the period of each flow stays the same but each flow has a dedicated WCAT for each criticality level. The proposed example can be easily transposed with dedicated periods for each criticality level, but we suppose the period as constant for readability issues.
In the topology of figure 8.1, we define a set of $\{v_1, v_2, v_3, v_4, v_5\}$ flows. Each flow v_i is described by the parameters which value are given in the table below. Basically, we consider flows without any offset (all flows are first activated at $t = 0\mu$s). We suppose that the switching latency of the physical links in the network is null ($sl = 0\mu$s).

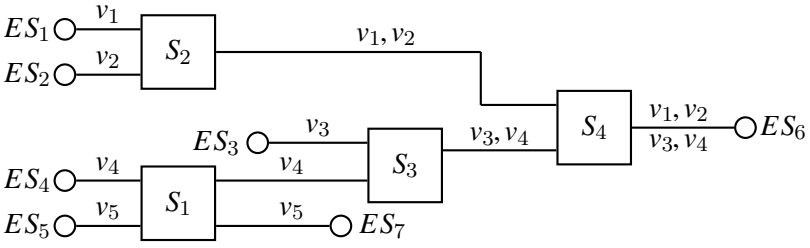


Figure 8.1: Network topology

v_i	$C_i^{LO}(\mu s)$	$C_i^{HI}(\mu s)$	$T_i(\mu s)$
v_1	10	20	50
v_2	10	-	50
v_3	20	30	80
v_4	10	-	100
v_5	15	-	60

The results are shown in figure 8.2. Conformly to what we presented for the centralized protocol, all messages from flow v_2, v_4, v_5 are dropped out from the network during the HI-critical phase (starting at $t = 50\mu$s). Concerning the flows v_2 and v_4, as they share in their path a switch in common with the HI-critical flow v_1, dropping them from the switches queues assures that they will not imply additional delay in HI-critical messages transmission.
In terms of transmission, we observe that the centralized protocol allows network nodes to transmit correctly $4\,(v_1), 1\,(v_2), 3\,(v_3), 0\,(v_4), 1\,(v_5)$ messages during the time interval. It represents the transmission of 100 % of the HI-critical traffic and $\frac{1}{4} + \frac{0}{2} + \frac{1}{4} = 20$ % of the LO-critical traffic. The global messages transmission rate of the example is evaluated at 41.17 %.
The observation of the transmission of flow v_5 clearly shows that the centralized protocol tends to ignore and drop non critical traffic, leading to pessimism in the quantity of messages managed in the network. That leads to our problem: how can we propose a MC management protocol which lowers QoS loss due to criticality changes?

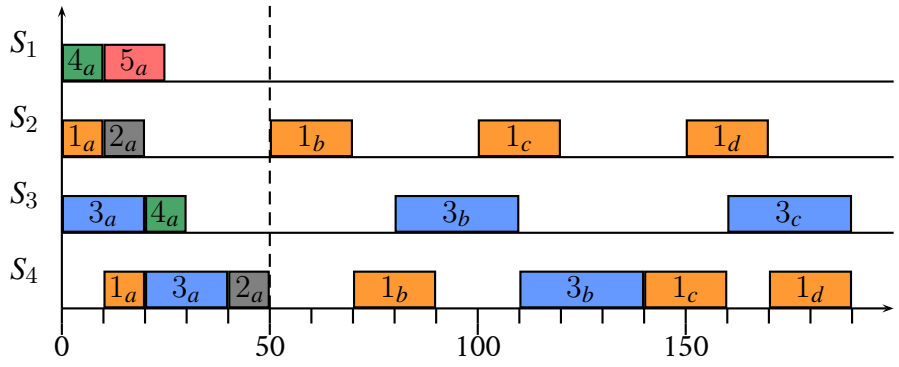


Figure 8.2: Scheduling messages

On the contrary, we observe that the LO-critical flow v_5 does not share any switch in its path with a HI-critical message. Its transmission does not impact HI-critical traffic, but the flow is still dropped out of waiting queues during the HI-critical phase.

8.3 Decentralizing the mixed criticality management

8.3.1 Concept

Based on the full-centralized and half-centralized approaches presented in chapter 7, the solution we propose is to let each node manage its own criticality level independently from other nodes. We want to propose a decentralized MC protocol where each switch will be responsible for managing its own criticality level, independently from the other switches. With this protocol, each switch is able to change its internal criticality level (depending on the same events as for centralized protocol), but there is no more global criticality synchronization in the network.
The solution is as follows: each switch gets its own criticality level, stored internally in the switch's memory. This level can be modified locally in the switch, depending on the received messages. If the switch receives a message marked as γ_{new}-critical, or when it detects that a message exceeds its WCAT for the current criticality level, the switch changes its local criticality level.
We consider that, as soon as a switch is in γ_{new} level, it will only transmit messages of γ_{new} level. In order to verify the timeliness of the protocol, we focus on the transmission delay of critical messages in the network.

8.3.2 Dual-criticality level

Protocol description

In terms of modelisation, we suppose a network $\mathcal{N}$, composed of a set of flow $\mathcal{V} = \{v_1, v_2, ..., v_n\}$. First, we suppose the integration of decentralized protocol in a dual-criticality level network. The two different criticality levels are denoted as $\{LO, HI\}$. Each flow is defined with a specific WCAT for each criticality level and we suppose that, for a flow v_i which does not belong to level HI, we have $C_i^{HI} = -1$. For each switch $j \in \mathcal{N}$, we note its local criticality level as Γ_j.
We use the Ethernet 802.1Q tag to associate a criticality level to each message (see chapter 14.1 for details on this implementation). In each switch of the network, we implement a message scheduler in order to

manage the criticality level of the switch and to filter non critical messages.

```
Data: Incoming message m_i, node criticality level Γ
Result: Message filtering, new Γ value
1  if Γ == LO then
2      if C_i > C_i^LO then
3          Γ_temp ← readCriticalityField(m_i);
4          if Γ_temp == HI then
5              send(m_i);
6              Γ ← HI;
7          else
8              // A LO message exceeded its LO-WCAT
                 reportError(m_i);
9          end
10     end
11     send(m_i);
12 else
13     Γ_temp ← readCriticalityField(m_i);
14     if Γ_temp == HI then
15         send(m_i);
16     end
17 end
```

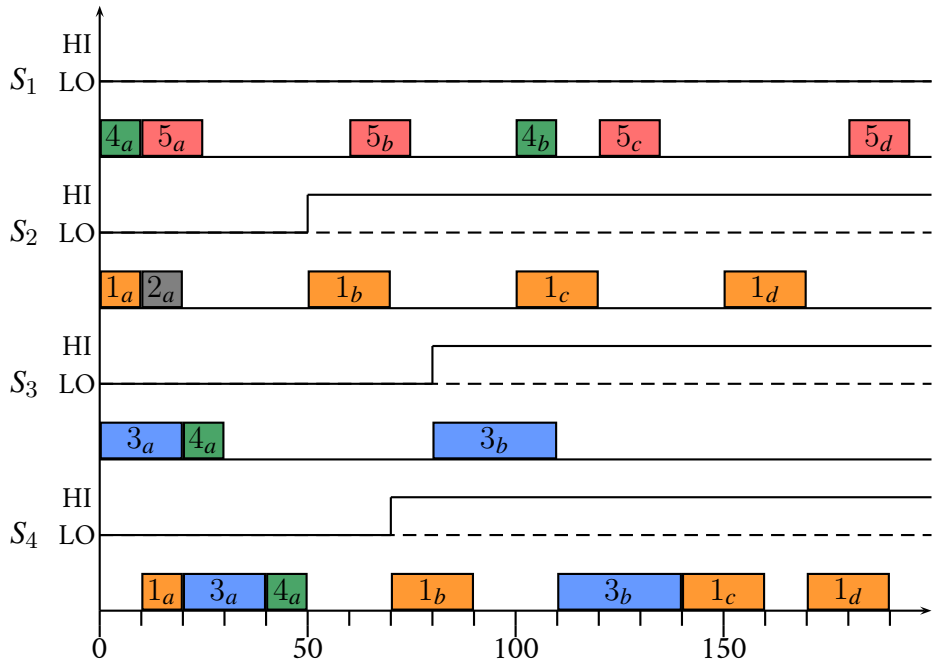
Algorithm 2: Decentralized MC management

The internal behavior of each switch is managed by a dedicated scheduling algorithm (see algorithm 2). The scheduling of messages in each switch depends on the current criticality level of the switch. In LO mode, the switch automatically detects LO-WCAT exceedings to change to HI level. On the contrary, in HI mode, the switch does not analyze a message WCAT but it directly reads its criticality field in order to decide whether to send the message or not.

This process allows the switches to transmit HI-critical messages even if their size is lower than the size corresponding to their LO WCAT. The criticality level change inside a network topology is nomore done at the same time for all switches, but happens at a specific date for each switch. Switches which are still on LO-mode can continue transmitting LO-critical messages, as long as they do not receive a HI-critical message. Moreover, switches which do not receive HI-critical traffic can stay in LO mode.
Algorithm 2 checks both the size and the criticality level of each message. As long as a HI-critical message has its real transmission delay lower than its LO-WCAT, a switch can transmit both LO and HI flows.
Thus, as the decentralized protocol does not rely anymore on a synchronization process, criticality level changes do not imply any delay in criticality changes despite the internal delay to change the criticality level inside a switch, which is common to both centralized and decentralized protocols. This delay depends on the electronical configuration of the switch and consists in reading and writing in the switch internal space. In our work, we consider this delay as integrated in the switching latency.

Example

In order to illustrate the impact of the decentralized protocol on LO-critical traffic, we focus on the example presented in section 8.2.1. We suppose the same topology, flows and parameters as previously defined, but we assume that MC in the network is managed with the decentralized protocol. We obtain the results described in figure 8.3.
The number of non critical messages which are transmitted is equal to: 4 (v_1), 1 (v_2), 3 (v_3), 1 (v_4), 4 (v_5), which represents a total of 76.4 % of guaranteed transmissions (100 % for HI-critical traffic, and $\frac{1}{4} + \frac{1}{2} + \frac{4}{4} = 60$ % of LO-critical traffic).



The decentralized MC management protocol allows switches to transmit LO-critical messages even when specific switches are on HI-critical level. As long as LO-critical messages path do not cross a HI-critical message path, their transmission will not be stopped. That is what we observe for messages from flow v_5, which are still transmitted as long as S_1 stays in LO-mode.

Figure 8.3: Decentralized protocol

Delay computation

In order to focus on the timeliness of this new decentralized protocol, we need to be able to bound the worst case delay of any HI-critical transmission. We propose here an evaluation of this delay using the Trajectory Approach.

Focusing on the end-to-end transmission delay of a message implies taking into account the impact of HI-critical traffic inside the network. In both protocols (centralized, decentralized), the delay induced by HI-critical traffic is computed the same way with the Trajectory Approach. It means that the impact of this traffic, and the additional transmission delay it represents, is the same for both protocols. This traffic has already been detailed previously (see chapter 7).

We assume that changing the criticality level inside a switch is instantaneous. As soon as a HI-critical message arrives in a switch, this switch changes its internal criticality level . It means that, as soon as there is a waiting HI-critical message in a switch, this switch will not have any additional LO-critical message waiting to be transmitted. Nevertheless, we can still have a non-preemptive effect induced by LO-critical traffic.

The computation of the worst case transmission delay of a HI-critical message m_i from flow v_i consists in computing the potential non-preemptive delay in each encountered switch along its path. If we suppose a HI-critical message m_i defined by $\{\vec{\mathcal{P}}_i, \{C_i^{LO}, C_i^{HI}\}, T_i\}$, the expression of the non-preemptive delay applied to end-to-end transmission delay of m is expressed as:

$$\sum_{n \in \vec{\mathcal{P}}_m} (\max_{n \in \vec{\mathcal{P}}_i}(C_i^{LO}, C_i^{HI}) \quad (8.1)$$

The non-preemptive effect can be induced by any LO or HI-critical traffic. If we focus on the expression of the switch-criticality delay presented for the centralized protocol (see section 7.2.4), we can see that the same non-preemptive effect is applicable to the transmission of the SCC message in the centralized approach.
The expression of this worst case end-to-end transmission delay is based on the hypothesis that there is no HI-critical traffic in the network. It is based on the assumption that HI-critical messages in HI mode can only be delayed by LO messages encountered along their path inducing a non-preemptive effect potentially induced by each LO-critical message, or by other HI-critical messages which can be with a higher priority.
Nevertheless, in order to establish a reliable worst case end-to-end transmission delay evaluation, we have to take into account HI-critical trafic and bound the delay induced by this additional traffic. We consider a FIFO scheduling policy. We obtain the following expression:

$$\sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} (1 + \lfloor \frac{t + A_{i,j}}{T_j} \rfloor) * C_j^{HI} \tag{8.2}$$

The global worst case response delay of flow v_i in the last node of its path $last_i$ is computed with the expression of Trajectory Approach, given in chapter 4. We obtain:

$$\begin{aligned} W_{i,t}^{last_i} = & \sum_{\substack{v_j \in \{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i \cap \vec{\mathcal{P}}_j \neq \emptyset}} (1 + \lfloor \frac{t + A_{i,j}}{T_j} \rfloor) * C_j^{HI} \\ & + \sum_{k \in \vec{\mathcal{P}}_i \setminus \{last_i\}} (\max_{k \in \vec{\mathcal{P}}_j} (C_j^{LO}, C_j^{HI})) \\ & + (|\vec{\mathcal{P}}_i| - 1) * sl \\ & - \left(\sum_{j \in \vec{\mathcal{P}}_i \setminus \{first_i\}} (\Delta_{i,t}^j) - t \right)^+ \\ & - C_i^{HI} \end{aligned} \tag{8.3}$$

If we consider the worst case end-to-end transmission delay of a HI-critical message which is the only HI-critical message in the network, it will only be delayed by LO-critical traffic. In that case, we obtain:

8.3.3 Changing criticality level back to LO level

In a dual-criticality level network, all HI-WCAT of HI-critical flows are necessarily equal or higher to their LO-WCAT. This hypothesis of the hierarchical structure criticality levels has been formulated in [114] and detailed in section 6.4.3. It is the same for the period, which is necessarily equal or shorter

during HI-critical phases. This hypothesis builds an order of importance among the criticality levels of a network topology.
Given this hierarchical structure, we defined the following property. As soon as the new criticality level γ_{new} is more important than the current one γ_{anc}, we immediately trigger a criticality level change. On the contrary, when γ_{new} is less important, we implemented a decision process based on a criticality table management in order to decide whether or not to change the criticality level in the network.
In the centralized protocol, this process of changing back to a lower criticality level is operated when each node of the network has not receive any critical message for a certain period of time. With the decentralized protocol, criticality level is not anymore managed through the criticality table stored in the central node. As a result, we have to propose an alternative to manage criticality changes to lower levels. The local criticality management can be used for this. Each node will be configured to be able to change back to LO level by itself.
We have to characterize the waiting delay before triggering this criticality level switch. We are working with sporadic and periodic flows. A pertinent evaluation of this waiting delay would be to compute it from the expression of the different periods (or minimum inter-arrival time) of each flow transiting through the switch. We propose to implement, in each switch, a waiting delay equal to the highest period of all flows in the network:

$$S_{LO} = \max_{\substack{v_i \in \mathcal{V} \\ C_i^{HI} \neq -1}} (T_i^{HI}) \tag{8.4}$$

We are sure that each periodic flow produced at least one message during S_{LO}. The delay S_{LO} is common to all switches in the network. This delay is reliable only for periodic flows. For sporadic flows, as we cannot guarantee a maximum inter-arrival delay between two messages, we cannot guarantee at least one message emission per flow during S_{LO}. It the same problem as we explained in 7.2.3: for sporadic flows, the only solution we can provide is to base S_{LO} value on minimum inter-arrival times.
Implementing S_{LO} value in each node implies waiting for a specific delay, computed from different flows which do not transit through the node. It can represent a potential loss of LO-critical messages: if the HI-criticality phase is too long, it represents an additional period of time during which there is no LO message transmission. In order to answer to this problem, a proposed solution is to define a specific waiting delay dedicated for each node k. This implies, for each node, knowing all the flows which transit through it (which is not necessarily the case in non-industrial networks, for example).
Spending time in HI mode while there is no HI message transmission represents a loss of ressources. If we want to improve the QoS by reducing this delay, we can define a delay indexed on the periods of the flows transiting through each node. This means that a node k will have to wait during a delay S_{LO}^k, defined as:

$$S_{LO}^k = \max_{\substack{k \in \vec{\mathcal{P}}_i \\ C_i^{HI} \neq -1}} (T_i^{HI}) \tag{8.5}$$

The problem represented by the expression of this delay is that it implies configuring each switch independantly. As this delay relies on the different flows which pass through the switch, it implies, for each switch, knowing the different flows which will transit through it.
As a conclusion, the decentralized MC management protocol proposes a MC management local to each node. The protocols allows the network nodes to increase and decrease their criticality level. This assures a bounded delay for HI messages transmission and guarantees isolation between LO and HI traffic.

8.3.4 Extension to multi-criticality levels network

The decentralized protocol has been introduced for dual-criticality level management, with networks based on LO (non critical) and HI-critical traffic. But, as shown, we often have to define more than two criticality levels (mission-critical, vehicle-critical, safety-critical for example). We propose to extend the decentralized protocol to multi-criticality levels networks.
We suppose the network composed of $\{\gamma_1, ..., \gamma_{k-1}, \gamma_k\}$ different criticality levels, with Γ_j the current criticality level in node j. The hierarchical structure among criticality levels formulated in section 6.4.3 allows us to assume that each incoming message $m_i = \vec{\mathcal{P}}_i, \vec{C}_i, T_i$ in node j can be considered with u different levels of criticality. Based on this assumption, we suppose an incoming message m of criticality level u incoming in node j, of current criticality level Γ_j. We obtain:

- If $u = \Gamma_j$, the node j stays in its current criticality mode and the message is transmitted.
- If $u < \Gamma_j$, the message m is dropped out.
- If $u > \Gamma_j$, we increase the value of Γ_j to u and the message is transmitted.

From this assumption, we deduce the MC management algorithm described in algorithm 3.

Delay computation

In a multi-criticality levels network, the worst case end-to-end transmission delay of a message depends on the current criticality level of each node. According to the hierarchical structure among criticality levels, when there is a conflict between different messages of different criticality levels, we always privilege the one with the highest criticality level. The transmission delay we will guarantee is the one for the message with the highest criticality level.
Inside a network $\mathcal{N}$, we can observe a mixed transmission of messages from various criticality levels. Each message can represent a potential additional delay due to non-preemptive effect. Only messages with a higher criticality level are likely to induce such non-preemptive effect (in worst case analysis), as messages with lower criticality level are dropped out from waiting queues.
We suppose a network $\mathcal{N}$ composed of a set of n flows $\{v_1, ..., v_{n-1}, v_n\}$, each flow characterized by $\{\vec{\mathcal{P}}_i, \vec{C}_i, T_i\}$, with $\vec{C}_i = \{C_{\gamma_1}, ..., C_{\gamma_{u_i-1}}, C_{\gamma_{u_i}}\}$. It means that each flow v_i belongs to a certain number of criticality levels u_i. We note k the number of possible criticality levels in the network, which means $\forall i \in [1; n], u_i \leq k$. We can express the worst case end-to-end transmission delay of a message i from flow v_i, of criticality level γ_{u_j} with the Trajectory Approach (see chapter 4, expression 4.16). We obtain:

The integration of this algorithm integrates multi-criticality levels management with the decentralized protocol. When we have an incoming message m, the node checks the size of m.
If this size corresponds to a greater size than the WCAT of m corresponding to Γ_j, then we need to increase the value of Γ_j. If not, it means that the message is of Γ_j-criticality level and needs to be transmitted.
In the case where the transmission delay is lower than the WCAT of Γ_j level, the node checks the criticality level of the message, in order to determine if the node can send the message or not.

```
Data: Incoming message m_i, node criticality level Γ_j
Result: Message filtering, new Γ value
if C_i > C_i^{Γ_j} then
    Γ_temp ← readCriticalityField(m_i);
    if Γ_temp > Γ_j then
        send(m_i);
        Γ_j ← Γ_temp;
    else
        if Γ_temp == Γ_j then
            send(m_i);
        end
    end
else
    Γ_temp ← readCriticalityField(m_i);
    if Γ_temp ≥ Γ_j then
        send(m_i);
    end
end
```

Algorithm 3: Decentralized MC management for multi-criticality levels

$$
\begin{aligned}
R_i(\gamma_{u_j}) &= \max_{t\geq 0}\{W_{i,t}^{last_i}(\gamma_{u_j}) - t + C_i^{\gamma_{u_j}}\} \\
W_{i,t}^{last_i}(\gamma_{u_j}) &= \sum_{\substack{v_j\in\{1,2,\ldots,n_f\} \\ \vec{\mathcal{P}}_i\cap\vec{\mathcal{P}}_j\neq\emptyset \;\; l\in[u_j;k]}} (1+\lfloor\frac{t+A_{i,j}}{T_j}\rfloor) * C_j^{\gamma_l} \\
&+ \max_{l\in[1;k]}(C_j^{\gamma_l}) \\
&+ (|\vec{\mathcal{P}}_i|-1) * sl \\
&- \left(\sum_{j\in\vec{\mathcal{P}}_i\setminus\{first_i\}} (\Delta_{i,t}^{j}) - t\right)^{+} \\
&- C_i^{\gamma_{u_j}}
\end{aligned}
\tag{8.6}
$$

This expression is an upper-bound of the worst case reception time of a γ_j-critical message o is the last node $last_i$ of its path $\vec{\mathcal{P}}_i$.

8.4 Conclusion

The presented decentralized approach is an alternative way for MC integration inside RT Ethernet networks. It relies on a distributed management of the criticality level, dedicated to each node in the net-

work.
This protocol implements independant MC maangement of each node in the network. It allows the network designer to reduce the criticality level management costs in terms of transmission time. Moreover, dedicating the criticality level management to each node allows the network to be independant from clock synchronization, contrary to the centralized approach.
The simulations we made confirm the assumption about the benefit of the decentralized approach (see chapter 12). Its integration does not rely on clock synchronization and induces a better QoS rate for LO-critical traffic. As a conclusion, it makes this approach better suited for industrial networks where QoS and usability have to be guaranteed. This concerns particularly domains oriented to public utilisation, where LO-critical functionalities (oriented to comfort, multimedia, etc...) are the most frequent. We can mention domains such as automotive, public transports or IoT.
Depending on the utilization context and network infrastructure, both approaches can be adopted for MC integration inside RT networks.

Part IV

Real-Time Networks simulation with ARTEMIS

Chapter 9

Real Time network simulation with ARTEMIS

"Un bâtiment doit réunir trois caractéristiques: un bon emplacement, des fondations sûres et une exécution sans faille."

"Three things are to be looked to in a building: that it stand on the right spot; that it be securely founded; that it be successfully executed."

– Johann Wolfgang von Goethe [131]

Contents

9.1 What is ARTEMIS?

9.1.1 Introduction

ARTEMIS is a network simulation tool. It simulates the transmission of messages generated for predefined flows. These messages are transmitted through a network topology. ARTEMIS proposes to the user to model a network topology (input and output points, switches, wires) and to simulate the scheduling of a flowset inside this topology during a given time interval. The internal structure of ARTEMIS and its functional goals have been presented in previous works [132], [133].
ARTEMIS is organized around several major development axis. The main development guidelines of ARTEMIS are described as follows:

- Modular: each part of the tool is an individual module which can be launched and used on a standalone version. This modularity allows us to use ARTEMIS in two different ways: Either it can be used in a global full version with interconnected modules, or individual modules can be used to perform specific functionalities.
- Web-oriented: There is a clear separation in ARTEMIS between the graphical interface (which operates as an XML files generator) and the simulation core. The graphical web interface allows the user to build and install distributable versions for group of users, including the advantages of the web context: no individual installation on each working terminal, easiness to update and maintain.
- Open-Source: ARTEMIS was mainly designed for educational and research purposes. In order to be easily shared among community of users, ARTEMIS has been chosen to be free and open-source. Thus, its modular structure makes it easier for external development teams to design and build new modules for the tool, and the Open-Source aspect reinforces this approach.
- Easy-to-use: We did consider usability and ergonomy as an additional approach to take into account during the development, particularly by integrating specific efforts in the design of an usable GUI.

In this chapter, we propose to fully detail the internal structure and functional perimeter of ARTEMIS.

9.1.2 Functional description

ARTEMIS contains, as major modules, a basic simulation kernel, a topology and message generator and a GUI. All these modules are interconnected through an API, written in XML. The first fundamental element of ARTEMIS is its simulation core. It is written in Java and its role is to compute the transmission time of each message, from each flow, transmitted in a network topology.
ARTEMIS network modelling is based on the representation of a network with three basic elements [133]:

- The nodes represent the network operators (switches, entry points). These are the physical devices responsible for sending, receiving (end-systems) or forwarding data (switches).

- The links represent the bridges between nodes. These allow the nodes to be connected between each other.
- The flows represent the data. Each flow produces messages and is characterized by a sender, a receiver, and a static-defined path of nodes between them. This static definition is an external constraint due to RT networks: each path of each message has to be statically defined. This is a necessary condition to assure the determinism of transmissions inside a RT network.

Figure 9.1 shows the global structure of ARTEMIS's modules. ARTEMIS is organized according to different layers: the core layer (simulation and computing data), the module layer (plugged to the core through XML files), the XML layer (used to model data) and finally the user layer which includes the GUI. In this layer architecture, the XML layer operates as an data interface (API) which models the different aspects of datas needed by other modules: simulation configuration, graphical parameters, timing analysis description, etc...
Additionnally to modular structure, this layer organization allows an easier access to each functionality when debugging: it makes the development work easier to act on a specific functionality or correct a specific problem in a feature.

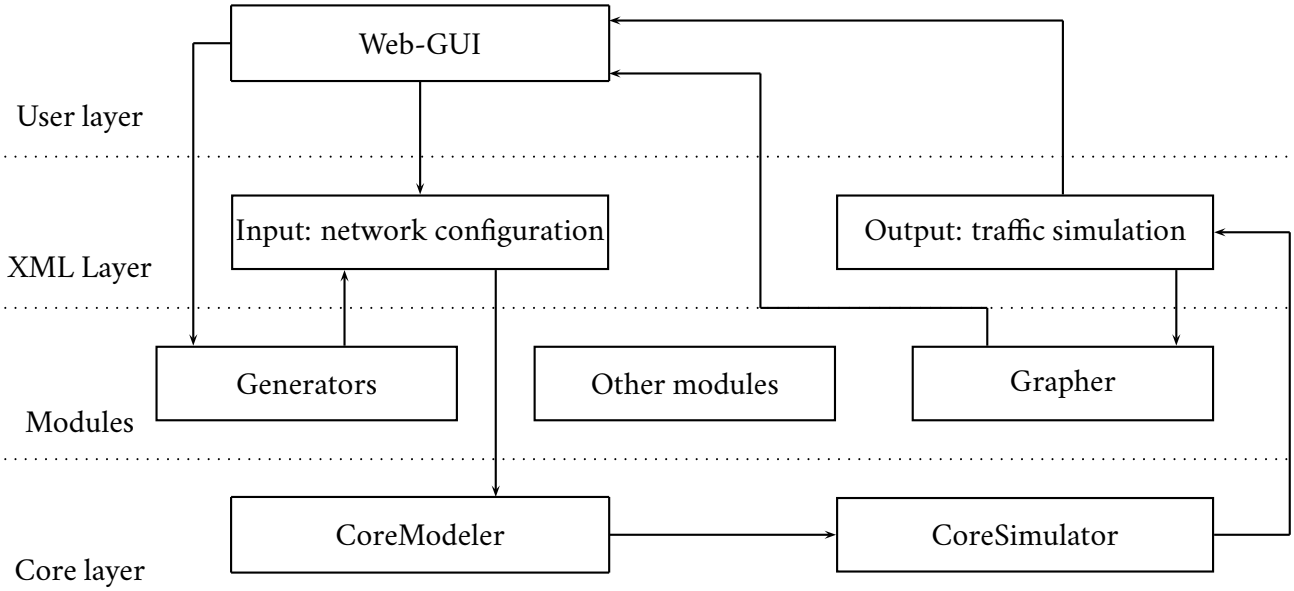


Figure 9.1: ARTEMIS functional structure

9.1.3 Software core

Core structure

The core of ARTEMIS consists of two different parts: the CoreModeler and the CoreSimulator. First, the CoreModeler is the entry point of ARTEMIS core. The role of the CoreModeler is to represent and model the network, according to the data saved in the XML files. The CoreModeler has to represent the network topology, messages, and all the network configuration. The CoreModeler is responsible for

parsing the XML input files and informations, building them as Java objects, in order to build a complete network simulation context: nodes, links and messages.
In terms of addressing, The CoreModeler is responsible for making the address attribution for all network nodes. Each node is identified by two different elements: the user-oriented identifier (defined in the GUI) and the network address (computed by the CoreModeler).
The modelling of the network done by the CoreModeler depends on the XML representation of the network built by the different modules of ARTEMIS. This XML representation is composed of different files which can either be manually edited by the user (or generated by external tools), generated through the GUI or automatically generated by ARTEMIS modules. These files are organized as follows:

- network.xml: representation of the topology (nodes and links). It also integrates bandwith representation. This file integrates the representation of the network structure, plus all the network platform description.
- messages.xml: it contains the complete representation of each flow in the network. Each message is described with the different criticality levels it belongs to. For each criticality level, we define the different parameters of each message (WCAT, path, period).
- config.xml: This file regroups all the parameters which are supposed to be applied globally on the topology or on the messages. This includes the MC management models (centralized, decentralized) and the transmission delay computation model (static, dynamic).
- graphconfig.xml: A secondary file used for graphical configuration and display modes in the grapher module. This file is not used by the core itself. It is used by the grapher module for display management.

The second part of ARTEMIS core is the CoreSimulator. Its role is to simulate the runtime phase of the network modeled by the CoreModeler. The CoreSimulator is responsible for simulating the time-oriented environment and for scheduling the flows transmission through the network topology.
The CoreSimulator bases its computation over the object structure built with the CoreModeler. The CoreSimulator produces XML output files describing the state of each node in the network, at each time instant of the simulation. In the output XML layer, each node has its own XML file, establishing a timing description of each node.
This split between CoreModeler and CoreSimulator has been operated in the beginning of the ARTEMIS design, in order to differentiate data parsing and modelling on one side, and timing analysis and RT scheduling on the other side. This allows us to build a logical process of simulation, which is detailed in figure 9.2.
In the work below, we detail the role and main structure of each configuration file, in order to detail the XML interface of ARTEMIS. The modularity and genericity of ARTEMIS are based on this file structure.

Network configuration

The topology description is based on a dedicated XML file. In this file, each node is characterized by a set of several user-defined properties:

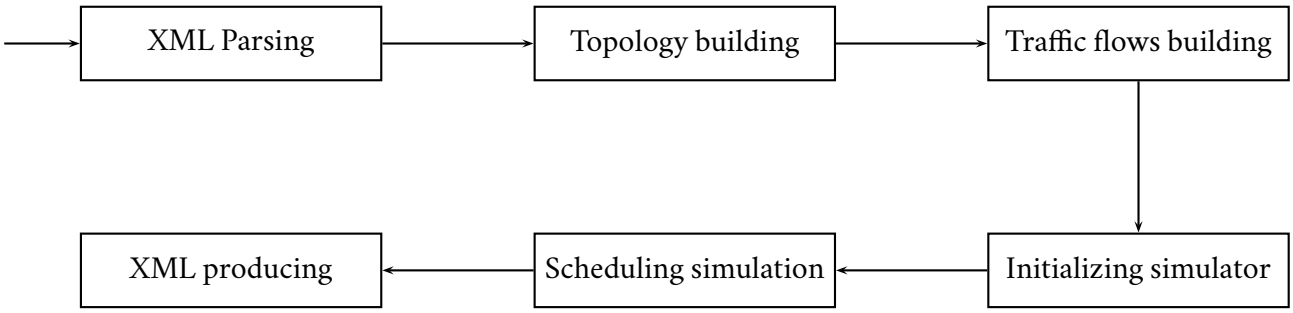


Figure 9.2: ARTEMIS core logic

```
<Network>
    <machine id="0" name="ES0" speed="1" shape="
    "><Config><name/></Config><Links><machinel id
    ="0"/></Links></machinel>
    <machine id="2" name="S1" speed="10" shape="
    "><Config><name/></Config><Links><machinel id
    ="2"/><machinel id="2"/></Links></machinel>
    <machine id="3" name="ES2" speed="1" shape="
    "><Config><name/></Config><Links><machinel id
    ="3"/></Links></machinel>
</Network>
```

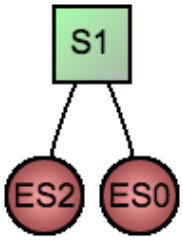


Figure 9.3: network.xml file example

- Name: This is the label of the node, used on the GUI.
- Rate speed: This is the data analyzing rate of the node, defined in Ms/s.
- Scheduling policy: the policy applied to the messages transiting through the node. Basic policies have been integrated in the ARTEMIS (FIFO, FP) and a generic interface allows the user to integrate additional ones.
- Outputs: the list of nodes the current node is connected to, as outputs.

An example of a network configuration file given in 9.3 shows the different description tags used in ARTEMIS XML formalization. Depending on the database structure, each node definition is linked to a unique id. Additional parameters (address management policy, for example) can be specified in the Config tag.
In ARTEMIS, we consider that each link between two nodes is oriented: if a node n_a is connected through an input link to a node n_b, flows through this link can only go from n_a to n_b. It means that if we want to emit data in the two directions between two nodes, the core will build two different links (full-duplex). As a result, we consider each node connection in the configured network as a full-duplex connection.

```
<Messages><message id="244"><criticality
    level="NC"><path>0,2,0</path><priority
    >0</priority><period>100</period><offset
    >0</offset><wcet>20</wcet></criticality
    ></message>
<message id="245"><criticality level="NC"><
    path>0,2,0</path><priority>0</priority><
    period>70</period><offset>0</offset><wcet
    >30</wcet></criticality></message></
    Messages>
```

v_i	$\vec{\mathcal{P}}_i$	C_i (μs)	T_i(μs)
v_1	$\{ES_0, S_1, ES_2\}$	20	100
v_2	$\{ES_0, S_1, ES2\}$	30	70

Figure 9.4: message.xml example file

Being based on an open well-shared standard like XML allows us to easily modify or add new informations in each file. Either by modifying a simulation configuration by manually editing the file or, for future uses, to modify the data representation model. This integrates genericity in the formalization of data, for example for benchmarking purposes.

Messages description

The messages description file contains all the needed informations to represent each flow likely to be integrated inside the core during simulation. Each message is produced by a flow and each flow of the simulation can be configured by the user. We consider in ARTEMIS that each flow is represented by a generic model: $\{\vec{\mathcal{P}}_i, \vec{C}_i, \vec{T}_i\}$. For each criticality level in the simulation, each flow can have a different WCAT and period. An example of a message.xml file is given in figure 9.4.
Similarly to the topologies, ARTEMIS integrates a flowset generator (detailed in chapter 11) to automatically generate flows for the simulation. This module will automatically generate the messages.xml file
Figure 9.4 shows that each message contains specific properties for each criticality level the network can manage. The structure of the file is to organize each message according to each criticality level it belongs to. For each criticality level, we define a set of properties $\{\vec{\mathcal{P}}_i, C_i, T_i\}$.

Platform configuration

Mainly, the parameters stored in the config file of ARTEMIS are dedicated to be used by external modules through a parsing phase. Timing analyze, flowset and topology generators, simulation pre-configuration are all simulation steps relying on this data modelling. An example of the file is given in figure 9.5.
The figure 9.5 shows an example of a global configuration file, specifying the duration of the simulation ($200\mu s$) and the electronical latency ($0\mu s$). We also note the different parameters used for transmission time computation: computation model (WCATmodel), computation rate (WCATrate) and MC management protocols (switch, protocol). These parameters are detailed further in 11. As a conclusion, this

Example of a messages.xml input file

```
<?xml version="1.0"?>
<Config>
    <time-limit>200</time-limit>
    <elatency>0</elatency>
    <WCATmodel>LIN</WCATmodel>
    <WCATrate>10</WCATrate>
    <switch>D</switch>
    <protocol>Decentralized</protocol>
</Config>
```

Simulation time	200 μs
Switching latency	0 μs
WCAT Generation	Linear (rate: 0.1)
MC changes	Dynamic
MC protocol	Decentralized

Figure 9.5: config.xml file example

config file is used for the global configuration of the simulation itself.

9.1.4 Time-oriented scheduling algorithm

The CoreSimulator is based upon a time-oriented algorithm, clearly differentiating each step of the process of message transmission. This algorithm is detailed in 4. At each instant of the simulation, we trigger an algorithm loop.

```
Data: network N, messages set S, scheduling
      policy S_p
Result: Scheduled traffic
for time ← 0 to limit do
    foreach machine in N do
        Generate new messages
        inputBuffer ← new messages
        message ← select(S_p, inputBuffer)
        analyze(message)
        if analyze ended then
            outputBuffer ← message
        end
    end
    foreach machine in N do
        Send(outputBuffer)
    end
end
```

Algorithm 4: ARTEMIS simulation algorithm

If we summarize, at each time and for each machine, the simulation algorithm is based on successive steps:

- Generate: for each end-system, we insert the potential new messages in the network coming from.
- Load: we load in input buffers all the incoming messages (generated or transmitted from other nodes).
- Analyze: We schedule the message and forward it.
- Prepare: Fully-analyzed messages are put in the output buffer corresponding to their destination.
- Send: All messages waiting in the output buffer are sent to the next node in their path.

Example

The simulation process of ARTEMIS core is based on the timing simulation of these successive steps for each machine during a specified time interval. We illustrate this algorithm on an example (see figure 9.6). This example is based on the configuration described previously. Flow v_1 is represented in orange, and flow v_2 in pink.

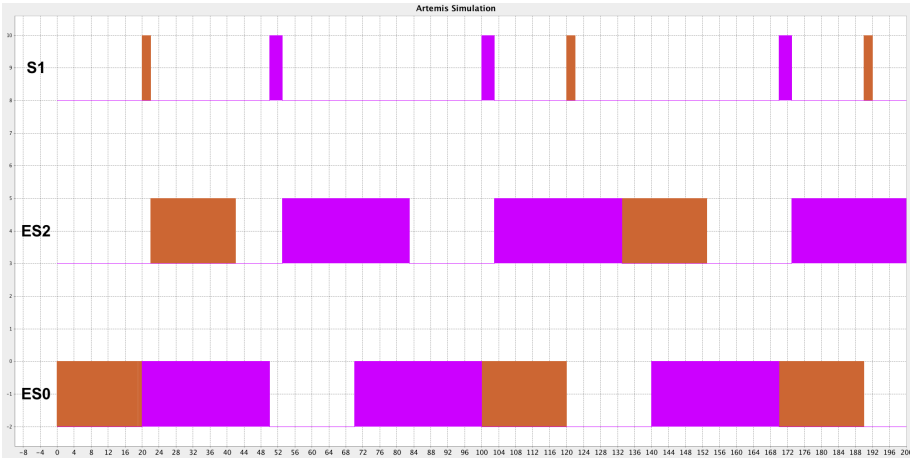


Figure 9.6: Artemis graphical results

The application of the scheduling algorithm described in 4 gives us the following process. At date 0 :

- Messages from ES_0 are generated (v_1^a for v_1 and v_2^a for v_2).
- Both messages are put in ES_O input buffer.
- We select each node which verifies : input buffer not empty, not currently transmitting a message. That only leaves ES_0.
- In each select node, we pick the message with the highest priority (v_1^a for ES_0). This message is loaded inside the node, and the node is marked as "currently transmitting". This will prevent the load of any additional message until the transmission is finished.

At date 1 :

- ES_0 is currently transmitting v_1^a. The transmission is not finished.
- S_1 and ES_2 are still empty.

etc... The recurrent application of the algorithm loop gives us the final scheduling simulation result described in figure 9.6.

9.2 Provided results

ARTEMIS provides timing analysis results about the schedulability of a given network configuration. We want to focus on the worst case end-to-end transmission time of each message in a given network

topology. This timing analysis allows us to detail also QoS evaluations, for example on the number of lost messages (in case of multiple criticality levels) and average latency induced by a node.
In order to detail these results, ARTEMIS provides them through two different ways:

- Graphical results: this consists in the graphical expression of the scheduling plan obtained by simulating during the defined time interval. The grapher module of ARTEMIS is responsible for establishing this graphical representation. An example of such graphical results is given in figure 9.6.
- Detailed results: ARTEMIS integrates a timing analyzer module which computes and proposes the transmission time of each message and computes the QoS guarantees offered by the network configuration. Through an XML file generated by the module, we can access to the detailed transmission analysis provided by the module.

Basically in ARTEMIS, there is a maximum number of 500 input and 500 output ports by node. This is a constraint used to limit the execution time of algorithms inside the core. ARTEMIS allows the user to create every kind of network, from small topologies with around 10 end-systems like in small home automation structures, to wide architectures like we can find in aircrafts systems or spaceship with more than 100 end-systems.

9.3 User Interface

The purpose of ARTEMIS is to provide an easy to use network simulation tool. It has to compute results in a limited time, with a detailed and easy to set up configuration process. In order to implementation this easiness, we present here the choices we made during the development to respect ergonomy and improve user experience.

9.3.1 Web-oriented architecture

As said, the GUI of ARTEMIS is based on a web architecture, which can be run by an Apache or any Hyper Text Transfer Protocol (HTTP) server implementation suite integrating a PHP compiler. Thes GUI of ARTEMIS is a PHP and HTML bridge between the user and ARTEMIS core. The web-oriented part for ARTEMIS is composed of set of PHP scripts triggering the main java functions of ARTEMIS kernel.
The user interface (web layer) in ARTEMIS is used to generate XML input files for the kernel and the different modules. The purpose of the GUI is to allow the user to quickly generate these files and then to make the ARTEMIS kernel run one or several simulations based on them.

Why this architecture?

The web-oriented architecture is the first step of ergonomy integration in ARTEMIS. User-friendliness is a major issue in ARTEMIS development. We wanted the tool to be easy to install and easy to spread among a group of users. Also, the tool has to propose a generic approach and to be easily adaptable to different kind of network simulation models, so to be adaptable to different kind of users. The client-server

architecture provided by HTTP servers allows the software to manage multi user concurrent access and utilization. As a result, a web interface allows to make ARTEMIS a distribuable simulator.
We wanted to extend the usability of the tool beyond the limits of development-familiar users. For example, we want the tool to be easy to present and use during pedagogical demonstrations. We picked web-oriented languages (CSS, HTML5). This allowed the development to integrate complex interaction and ergonomy functionalities (animations, dynamism) without specific complexity in the development itself. This represent a strong contrast with classical graphical-oriented libraries such as OpenGL or JavaFX, which require dedicated development skills. Thus, providing a GUI based on specific libraries would have implied to install additional pre-requisites on each working machine, implying problems of accessibility.
Also, web-oriented structure induces easiness of installation. Thus, as the kernel of ARTEMIS relies on Java, web tools and Java make ARTEMIS portable and installable on various operating systems. Eventually, required tools are free and easy to install. We made a public version (recherche.ece.fr/artemis/), which can be used without installation or infrastructure. Through multi-user managing, each connected user can manage its own list of different simulations.

Architecture details

The web architecture of ARTEMIS is based on the management of identification and simulation keys, each one corresponding to a given context and user. Each user can have one or several simulation keys, and all the simulation keys are stored in a central database (see figure 9.7). At each simulation, we give the kernel the simulation key, corresponding to a given set of input xml files.
The simulation identification works as follows: First, associated with the PHP session id of each user is a key manager (on the server side) which generates a simulation key for each new simulation associated to a session key. It means that each user is identified to the server with a unique id, derived from his session id. Thus, each couple (session id, simulation id) identifies a simulation in a unique way. Using this unicity, we can deduce the simulation data from its id.
When the user wants to launch a simulation, we sent the simulation id to the kernel. This id is linked to the simulation XML files. At the end of the kernel simulation, generated xml files are re-associated with the simulation id, and so with the user. This allows the GUI to find and parse the output XML files corresponding to the simulation triggered by the user. This provides a multi-user usability for ARTEMIS and guarantees data isolation among simulation parameters. It also leans two users cannot work on the same simulation, they will necessary be isolated (even in the same server).
The id association and management in the GUI allows to integrate an import and export solution. This allows the user to extract simulations and their results, either by transferring them to another implementation of ARTEMIS, or to exploit them with external tools (for example, for benchmarking purposes). All XML files related to a simulation can be automatically integrated into an archive in order to be exported. This function also allows the user to keep simulations configurations even when session id (used for user identification) are no longer valid. This is shown in figure 9.7.

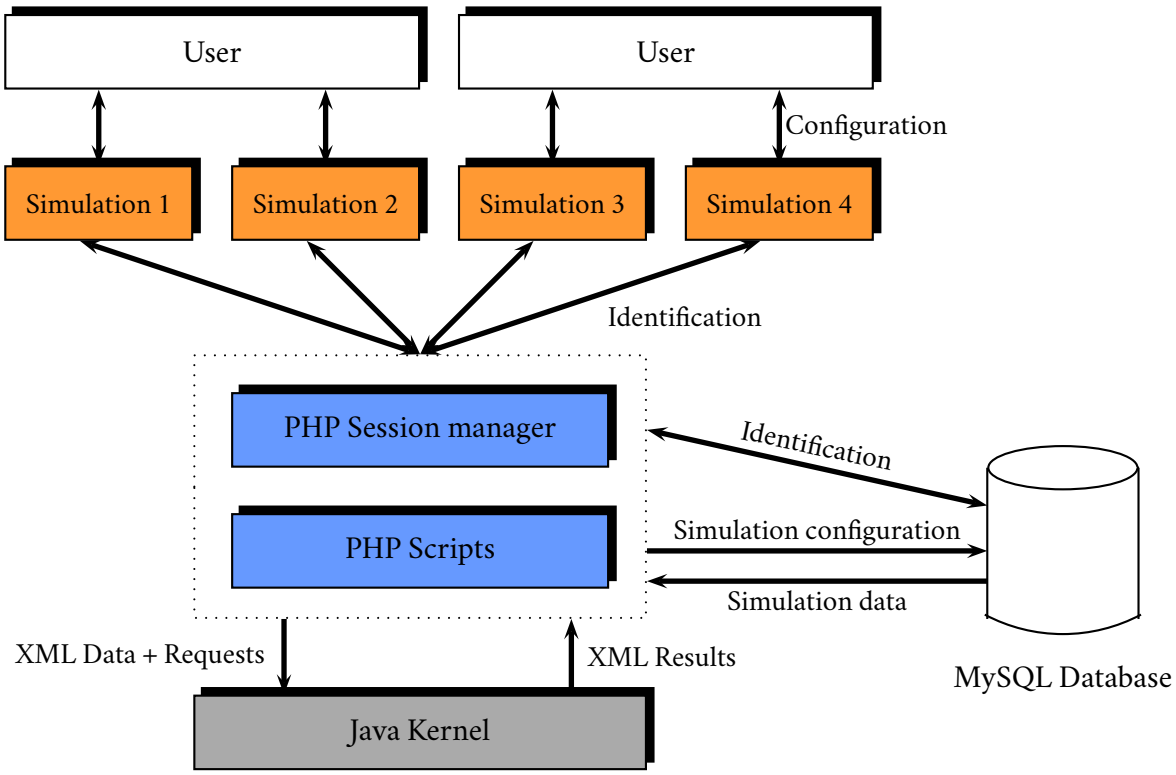


Figure 9.7: ARTEMIS web architecture

A few words about design

On the client side of the web application, all the development has been made with HTML5 + CSS3, in order to stay compliant to web standards. These languages are well recognized and interpreted by recent web browsers. For more details about CSS recognition and interpretation, see [134].
In terms of graphical design, ARTEMIS has been built as a classical software, with a centralized window regrouping all functionalities. Starting from this central window, we can access all the functionalities of the tool very easily through a tabs-structured organization.

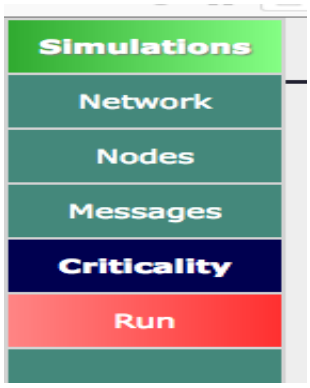


Figure 9.8: ARTEMIS GUI Tabs

The tabs described in figure 9.8 are defined in order for the configuration of a simulation to be sequential: first, we define the network topology by creating a set of nodes (switches, end-systems) interconnected through links. The GUI clearly points out the different between a switch and an end-system, in order to the global figure of the network to be easily understandable. Then, we continue with controlling and eventually modifying the links. Then, the process guides the user to define the different flows of the network (manually or automatically generated) and to specify their parameters: period, path, WCAT, etc...

The criticality management can be done through a dedicated tab, allowing the user to define the differ-

ent criticality levels of the simulation. For each defined level, the created flow will be in waiting to be associated to an eventual WCAT corresponding to this criticality level.
This sequential creation corresponds to the logical network definition process: first we define the model, and then we specify the implementation details. Finally, we run the simulation and detail the results. This respects the user approach when conceiving a network scheduling problem: starting from the general modelling and progressively iterating until obtaining a properly defined simulation context.

9.4 Conclusion

ARTEMIS is a RT network simulation tool, designed mainly for non-preemptive distributed flows scheduling through statically-defined network topologies. The functional structure of the tool has been clearly separated in different parts: user-oriented functions, simulation functions, results analysis functions. Dedicated module have been designed for each one of these class of functionalities. The integrated module are various: topology and flow generator, GUI, timing analyzer.
ARTEMIS has been mainly defined for specific RT network context: MC integration scenarios (see chapter 10). The functional structure of ARTEMIS allows the user to quickly define basic simulation results and obtain detailed results without requiring massive detailed specifications.
Eventually, in this chapter, we detailed the main architectural and design aspects of the tool, in order to present it from a functional point of view. In the following chapters, we will detail further specific aspects of ARTEMIS, especially the aspects of MC integration inside RT network simulation contexts.

Chapter 10

Integrating MC in ARTEMIS

"Bien que vous puissiez aimer ce que vous ne maîtrisez pas, vous ne pouvez maîtriser ce que vous n'aimez pas."

"Though you can love what you do not master, you cannot master what you do not love."

– Mokokoma Mokhonoana [135]

Contents

10.1 Introduction

ARTEMIS has been designed to focus on the simulation of MC integration scenarios inside RT networks. The functional perimeter of ARTEMIS implies for it to propose a solution to analyze the impact of criticality mode changes in RT networks, particularly in terms of traffic management. We want to show here the models we designed to answer to this problem. We detail these models in two steps: First, we present them and how we implement it in ARTEMIS core. In a second time, we run different simulations in order to prove the reliability of the implementation of these models.

10.2 Criticality management integration

10.2.1 The Criticality Manager

The MC management and the integration of criticality level changes during simulations has been implemented in a dedicated submodule of ARTEMIS. This submodule is a part of the CoreScheduler and is called the CriticalityManager. Its role is to be responsible for criticality changes integration in the topology and to be aware of the criticality level each node wants to change to. In the CriticalityManager, each node is attached to a current criticality level.
The CriticalityManager triggers the criticality changes (according to the configuration and to the runtime events). It assures the link between the CoreScheduler and the entity responsible for traffic management and transmission time computation. In terms of structure, the CriticalityManager is integrated in the CoreScheduler. When generating messages, all input and output informations transit through the CriticalityManager.

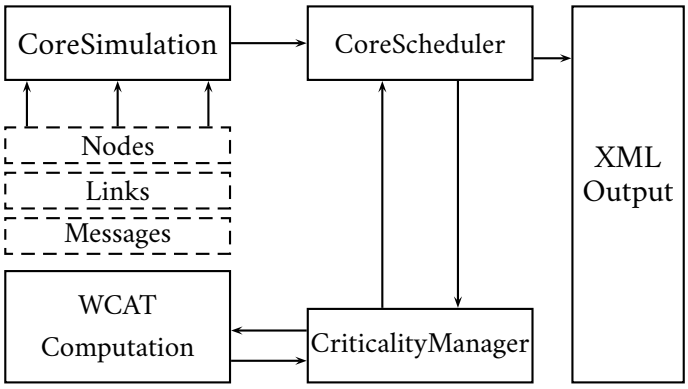


Figure 10.1: CriticalityManager integration in ARTEMIS

The CriticalityManager analyzes the different transmission times of messages to determine their impact in terms of criticality management (criticality table update, criticality level switch). The CriticalityManager also stores all the static defined criticality changes configured by the user.

10.2.2 Criticality table

When defining centralized MC management protocol (see chapter 7), we introduced the concept of criticality table. This is a memory space used to store the criticality table each node is ready to switch to. This is not used in the case of a criticality level increase (the decision is instantaneous in this case). On the contrary, the criticality table is used in order to trigger criticality level decreases in mixed critical RT networks.

In order to respect the protocols implemented in previous chapters, we need to define a model to represent the criticality table in ARTEMIS. This table does not indicate the current criticality level of each node (as this information is centralized, this value is the same for all nodes) but it specifies the criticality level the node wants the whole network to change to.
The criticality table is integrally managed by the CriticalityManager. When a node does not get a message from the current criticality level during a certain period of time, it updates its state in the criticality table, mentioning that this node is ready to change back to a lower criticality level. Once the criticality table indicates that all nodes are ready to change to a lower criticality level, the CriticalityManager orders a switch back.
The level to change to is not necessarily the same in all nodes (in case of a network managing more than 2 criticality levels). In that case, we pick the most critical level waiting among all nodes, and we reset the waiting timer for all nodes.

10.2.3 Criticality switch delay

The CriticalityManager is also responsible for integrating and managing the computation of each transmission time of each message, depending on the selected computation model. It is linked with a transmission time computer, allowing the CriticalityManager to filter messages depending on their criticality level, and associate a criticality level to each message depending on its transmission time.
The criticality switch delay is the combination of the SCC message transmission and reliable multicast delay (see the details of this expression in chapter 7). The computation of this delay is based on the determination of the network central node and the computation of the network depth (the longest possible path between the farest node and the network central node). This supposes a network with no loop.
According to the topology of the network and to the flows parameters, this criticality switch delay is automatically computed.

10.3 Message generation

10.3.1 Worst case and real case analysis

In ARTEMIS, when simulation a network scheduling scenario, we generate messages from the different flows we defined during the configuration phase of the simulation. Each one of these flows is characterized by a set of parameters: WCAT, period, path. For each criticality level, a flow has a dedicated WCAT which can be equal to -1 (the flow does not belong to this level) or to a specific positive value.
During configuration, we define each WCAT value for each criticality level of each message. At runtime, we can make two hypothesis. Either, each message analyzing time is equal to a WCAT (worst case hypothesis) or we suppose that the analyzing time of a message can be lower than the WCAT corresponding to the current criticality level (real case hypothesis). This second hypothesis introduces solutions to compute a message analyzing time, based on the values of its differents WCAT. This solution allows ARTEMIS to propose different delay computation models, like it was presented in SimSo [84].
When we configure a flow v_i, we define $\vec{C_i}$ which is the vector of its different WCAT (for each criticality level). . The different WCAT builds a threshold mechanism. At runtime, each time the flow v_i generates

a message, this message is characterized by an analyzing time C_i. If the current criticality level is Γ, the two hypotheses we made correspond to :

- Worst case hypothesis : For each message, we have $C_i = C_i^{\Gamma}$.
- Real case hypothesis : For each message, we generate $C_i^{best} \leq C_i \leq C_i^{\Gamma}$, with C_i^{best} the Best Case Analyzing Time (BCAT) of the flow.

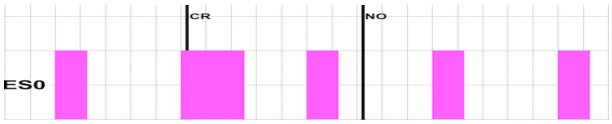


Figure 10.2: WCAT-based model

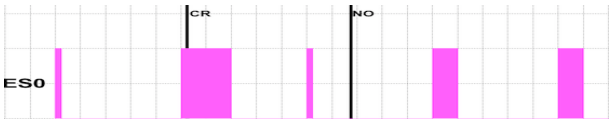


Figure 10.3: Real transmission time model

We ran a simple simulation example with a single flow v_i (for both real and worst case modellings). The flow v_i was configured in a dual-criticality (LO, HI) level network, with $T_i = 20\mu s$, $C_i^{HI} = 10\mu s$ and $C_i^{LO} = 5\mu s$. we obtained the results in figure 10.2 and 10.3. We set the BCAT of the flow to 0 for simulation purposes.

In the example of figure 10.3, the flow v_i produces 5 different messages (called m_1, m_2, ..., m_5). The results shows that, in worst case model, each individual transmission time is rounded to the corresponding threshold. With worst case modelling, we have $C_i = C_i^{LO}$ for m_1, m_3, m_4, m_5, and $C_i = C_i^{HI}$ for message m_2. With real case modelling, we have a different value of C_i for each message. We can guarantee that $0 \leq C_i \leq C_i^{LO}$ for m_1, m_3, m_4, m_5, and $0 \leq C_i \leq C_i^{HI}$ for m_2.

10.3.2 Criticality changes detection

In order to change the criticality level of the network nodes during simulation, we propose two different MC management models: static, and dynamic. Static management means that all the criticality changes are statically defined by the user before runtime. Dynamic detection implies for ARTEMIS to be able to detect whether a message exceeds its WCAT for the current criticality level, and to trigger a corresponding criticality level switch. We propose to detail both these models below.

Static criticality management

The static criticality management models rely on the centralized MC management protocol (see chapter 7). It is based on proposing to the user to manually define each date at which a criticality level happens. The user can define the date and the criticality level all the nodes in the network are supposed to change to. This can be defined through the GUI, or directly in the XML files layer.
This model limits the highest WCAT of each message, depending on the current criticality level. If the current level is LO, not any message will be able to have a transmission time higher than its LO-WCAT. Also, this model supposes that the criticality level switch is instant: there is no additional delay between the criticality switch order commanded by the user, and the effective criticality switch.

Dynamic criticality management

Dynamic criticality management is implemented in ARTEMIS in order for the core to decide by itself whether it has to change the criticality level of the network or not (depending on centralized or decentralized MC management protocols). But, implementing this solution implies for ARTEMIS core to integrate models to generate transmission times which will potentially exceed the WCAT of a message, corresponding to the current criticality level. In order to answer to this problem, we defined different time transmission computation models.
All the proposed models are based on the hypothesis that a message m is defined with several parameters: a WCAT for each criticality level it belongs to (minimum 1), and a BCAT. The BCAT represents the lowest transmission time a message can have. Usually, this transmission time is defined with the Ethernet standard size limit. It considers that the data field of a message is at its lowest size ($46bytes$). This supposes a minimum size of 64 bytes for each message. The minimum WCAT can be computed from this size, depending on the network global bandwidth.

10.4 Transmission time generation models

10.4.1 Uniform model

The uniform model of transmission time computation supposes a uniform distribution law of the different transmissions time of a message, in a interval bounded by the BCAT and the highest WCAT of the flow. The uniform model supposes that all potential transmission times in the interval are equivalently probable.
The probability distribution and cumulative functions are described in the figure 10.4. This is the representation for a message with a best transmission time of 0,61 μ s, and a WCAT of 7 μ s. This is computed on a 100 Mb/s global bandwidth.
In order to configure more accurately the model and to propose different transmission time generation solutions, the transmission time repartition interval can be reduced in its size. Its upper bound will be always the highest WCAT of the message, but the lower bound can be modified by selecting only 90, 80, ..., 10 % of the interval. Adjusting this rate allows the user to reduce the variability of generated transmission times.
During successive generation of messages from the same flow, each possible value of the transmission time (indexed on the time granularity of the simulation) will tend to be represented with the same number of occurences as all others. The uniform model is a fundamental to integrate in ARTEMIS, but the linear cumulative distribution tends to be naive in certain corner cases. In order to represent more specific cases and to hilight specific values of transmission times, we propose to detail another model.

10.4.2 Gaussian model

The second transmission time generation model integrated in ARTEMIS is based on a gaussian repartition law. It allows ARTEMIS core to generate different transmission times in the same interval between best transmission time and WCAT. This generation is based on a gaussian repartition of all the potential-

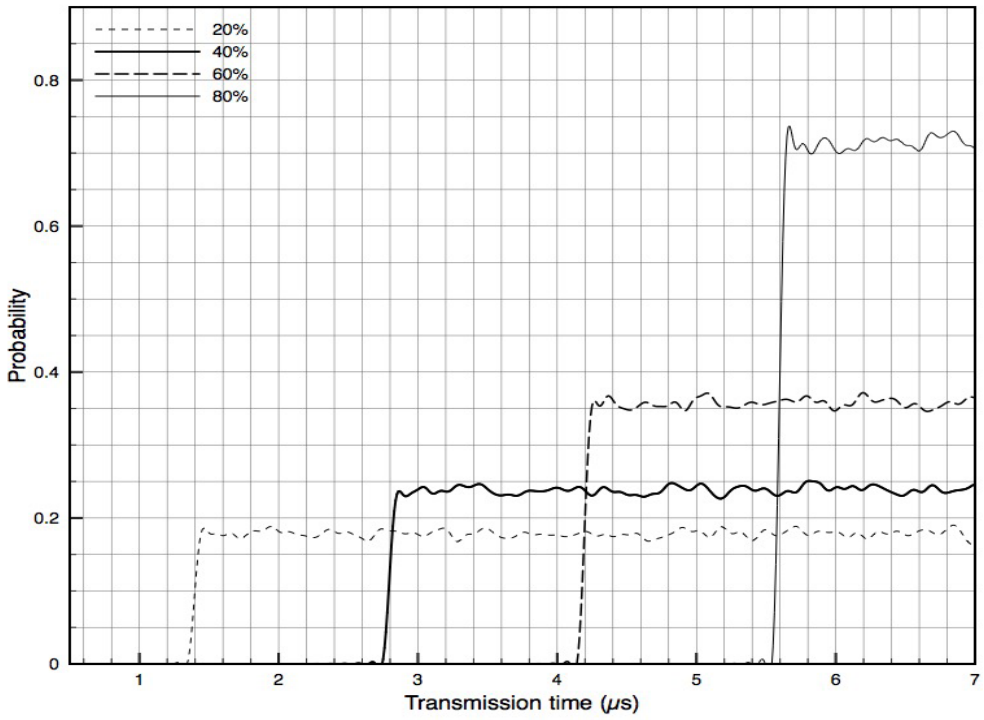


Figure 10.4: Transmission time generation / Uniform model

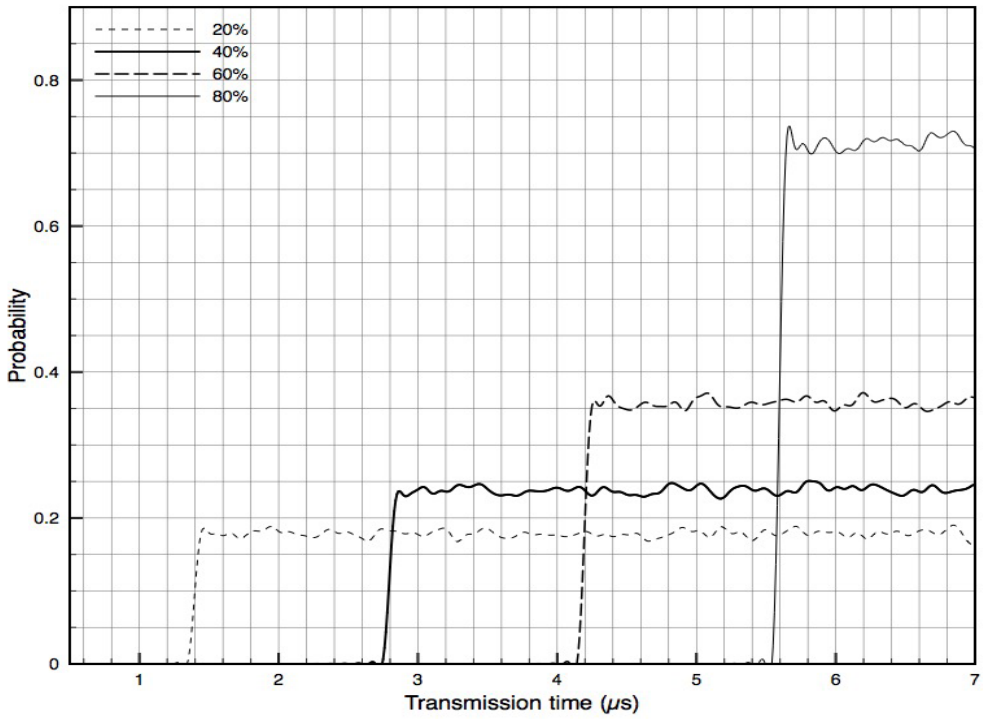


Figure 10.5: Transmission time generation / Uniform model, cumulative probability

transmission times.

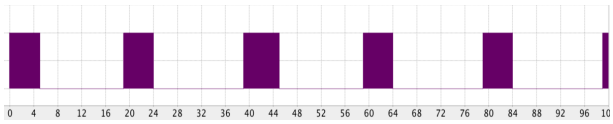


Figure 10.6: Uniform generation model

This gaussian model is centralized around the middle of the interval (BCAT, highest WCAT). The deviation of the model can be modified from 0.2 to 0.8 in order to adapt the gaussian model to the generation needs. We obtain a generation model like described in figure 10.7 detailed below.

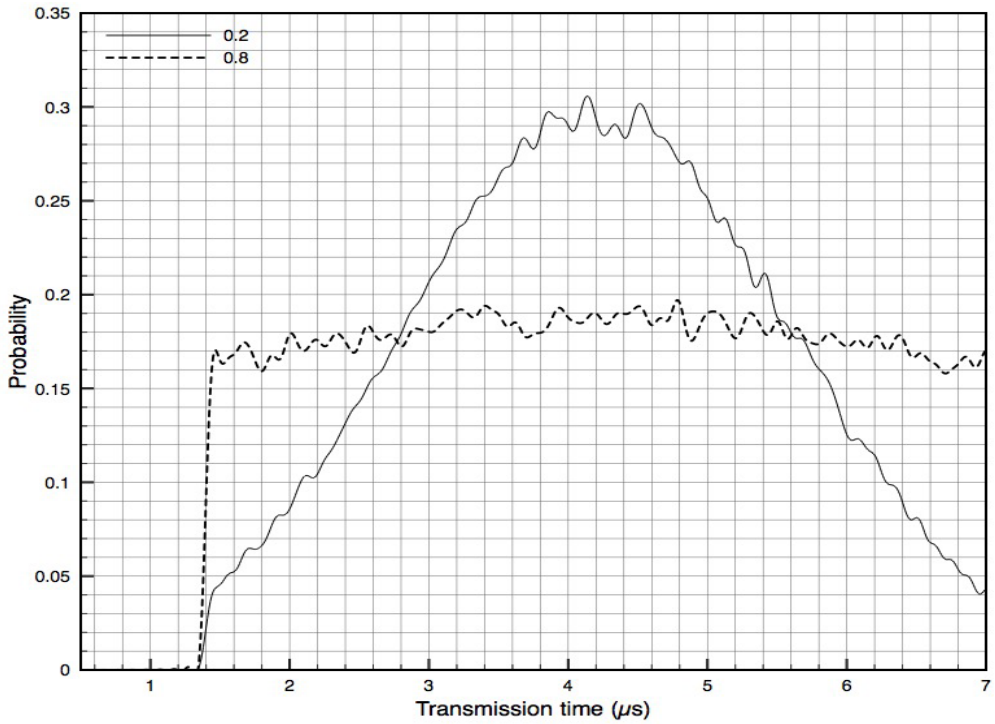


Figure 10.7: Transmission time generation / Gaussian model

We observe that a gaussian model with a high deviation (> 0.85) can give results comparable to a uniform model in terms of repartition. The figure 10.9 shows an example of gaussian generation model on the same flow.

In terms of representation, the gaussian model allows us to take consideration of concrete physical situations where a message tends to be around a specific value: for example, due to enconding reasons, two close values of speed (5.555 and 5.5551) can be encoded on a different number of bytes, but represent a similar situation.

To adapt the gaussian model to various situations, we propose to the user to be able to modify the more frequent value. These alternative models are based on the gaussian model, and they are detailed below.

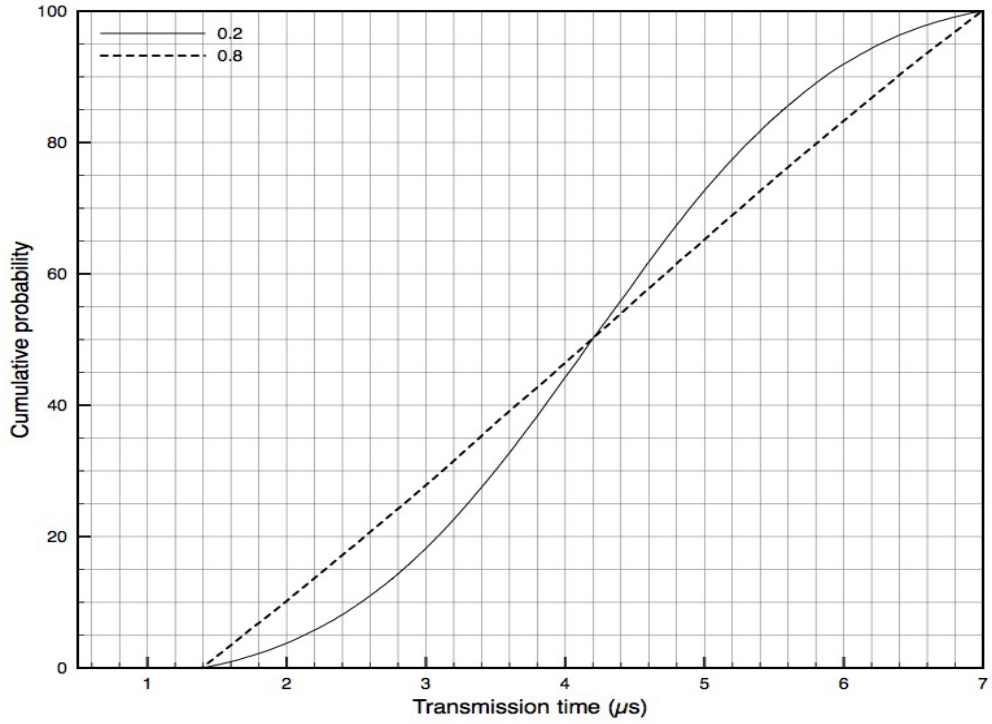


Figure 10.8: Transmission time generation / Gaussian model, cumulative probability

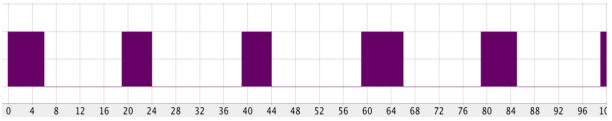


Figure 10.9: Gaussian generation model

The gaussian model privileges the values of the WCAT which are centered around a specific value c. The farest a value is from c, the lowest will be the number of occurences it will get.

Anti-Progressive gaussian

High values of transmission time corresponds to specific utilisations, due to critical cases for example. During the utilisation of a network, we can suppose that these situations do not represent frequent cases but, on the contrary, are due to isolated and punctual situations. It means that, during most of utilisation cases, the transmission time of a message tends to correspond to non critical mode, close to its BCAT (optimal situation). The anti-progressive gaussian model tends to support this assumption.
Anti-progressive gaussian is based on the same modelling as classical gaussian model, but the most frequent transmission time is supposed to be the message BCAT. The distribution function of the progressive model is given in figure 10.10.

Progressive gaussian

As a conclusion, the generation of transmission times in ARTEMIS integrates different distribution models. This allows the core to be compliant to various simulation contexts and to represent different network configurations. When generating random traffic for simulated network topologies, these generation models can also be used as traffic shapers by modifying the generation model attached to the

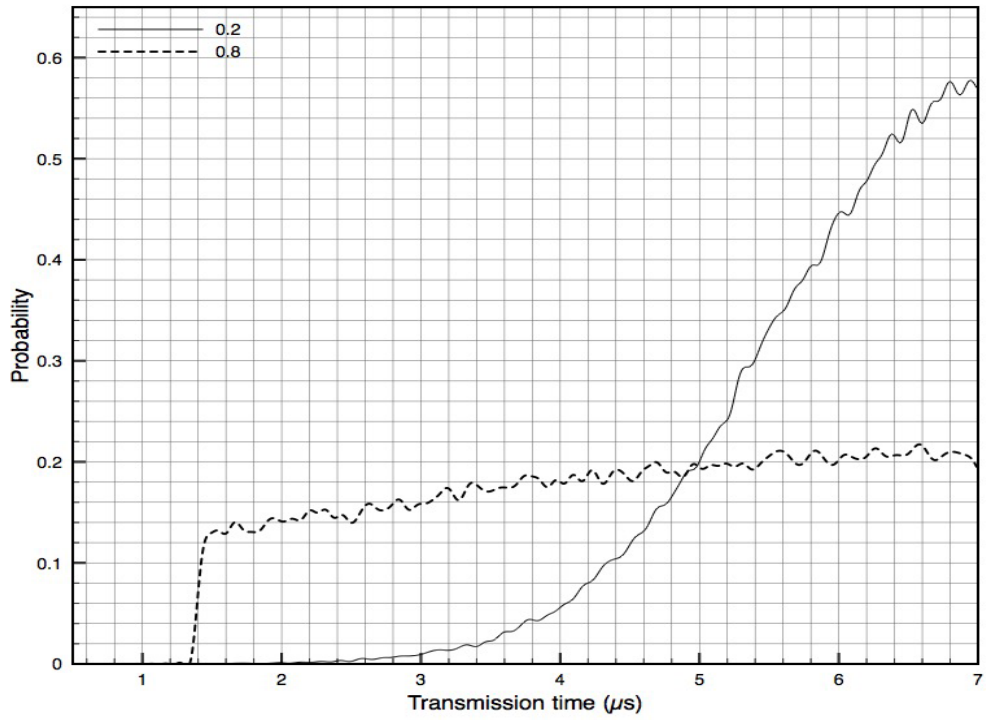


Figure 10.10: Transmission time generation / Anti-progressive gaussian model

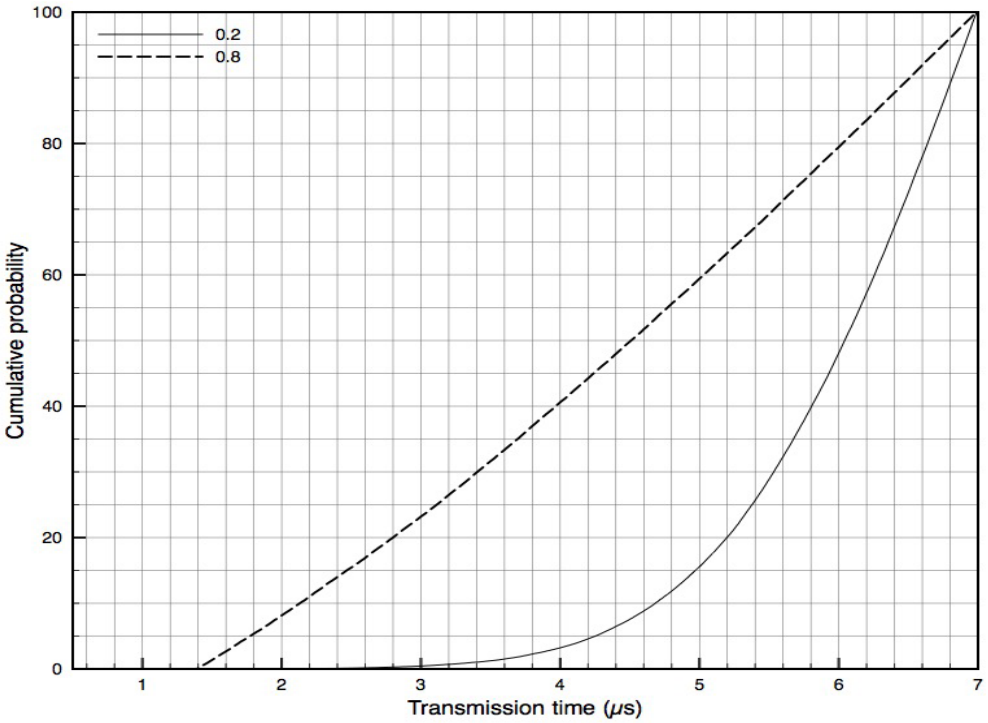


Figure 10.11: Transmission time generation / Anti-progressive gaussian model, cumulative probability

messages.

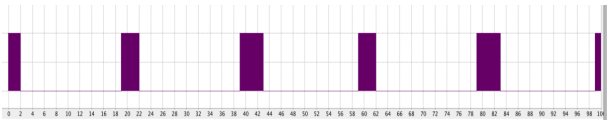


Figure 10.12: Anti-progressive gaussian generation model

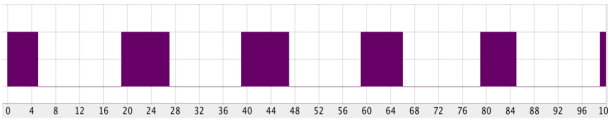


Figure 10.13: Progressive gaussian generation model

The figure 10.13 shows that the anti-progressive model tends to privilege the lowest possible values when generating a transmission time. This tends to maintain the system in lower criticality modes.

At the opposite of the anti-progressive model, the progressive model has been designed for simulation purposes. It tends to privilege the highest possible values when generating a transmission time. This is used, for example, when focusing on the resistance of a network to high-critical loads.

10.5 Simulation

10.5.1 Centralized approach

In order to model MC management inside ARTEMIS, we based our work on the MC management protocols we previously presented in part III. As a conclusion, we integrated two different solutions to manage MC inside ARTEMIS: the centralized and decentralized method.
As shown, the criticality management integrated in ARTEMIS is based on static or dynamic MC management. We simulated both these modes with the centralized approach.
First, we start with the static centralized model. It is based on the hypothesis that the MC in the network is managed according to the centralized protocol of MC management in networks. The criticality is globally managed as a centralized information and is shared in all the nodes of the network through a reliable multicast protocol. Additionnally, during a specific criticality level, all messages which do not belong to it are dropped out.
We consider that all criticality changes are statically defined by the user. In this model, we do not consider events such as SCC messages or WCAT-exceeding detection. Once a MC level has been defined at a specific instant, its implementation is instant. Thus, the static implementation supposes that all criticality level changes are user-designed during system design. The user defines, for each criticality level switch, the exact time at which it occurs and the level to change to. The network and the algorithmic core of ARTEMIS cannot change or modify these informations. It means that there is no switch criticality delay induced by the platform at runtime in static MC management mode.

Static Example

In order to illustrate the static centralized management in ARTEMIS, we built a simulation example. The purpose was to present a simple topology with a small set of flows to clearly identify the reliability of the protocol and to understand its way of implementation in ARTEMIS core.
In this simulation, we used the ARTEMIS topology builder to implement a simple topology as described in figure 10.14. This topology is composed of 4 switches and 5 different end-systems. We consider basically that S_3 is the central node of the network.

v_i	$\vec{\mathcal{P}}_i$	T_i (μs)	$C_i^{LO}(\mu s)$	$C_i^{HI}(\mu s)$
v_1	$\{ES_0, S_1, S_2, S_3\}$	30	5	8
v_2	$\{ES_1, S_1, S_2, S_3\}$	40	4	7
v_3	$\{ES_2, S_2, S_3\}$	40	2	-
v_4	$\{ES_2, S_2, S_3\}$	30	4	8
v_5	$\{ES_3, S_4, S_3\}$	40	5	8
v_6	$\{ES_4, S_4, S_3\}$	50	9	-
v_7	$\{ES_4, S_4, S_3\}$	40	1	4
v_8	$\{ES_5, S_4, S_3\}$	40	2	-

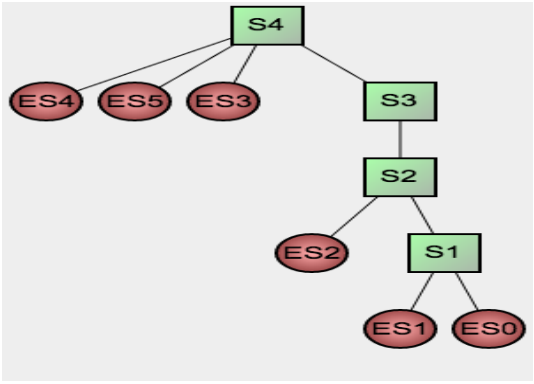


Figure 10.14: Topology example

We built a network with two different criticality levels (LO, HI), with considering that $\forall i \in \mathcal{N}, C_i^{LO} < C_i^{HI}$ for all HI-critical flows. On the contrary, we consider $C_i^{HI} = -1$ for all LO-critical flows. In this simulation, we performed a worst case analysis (each transmission time is rounded according to the threshold mechanism of WCAT).

In terms of criticality management, we set up two different criticality changes. First, we suppose an increase of the criticality level from LO to HI at $t = 50\mu$s. Secondly, we set a criticality level decrease from HI to LO at $t = 150\mu$s. Considering the topology, the flows and the different criticality parameters, we set it up into ARTEMIS and ran the simulation. We obtained the results described in the gantt chart of the figure 10.15.

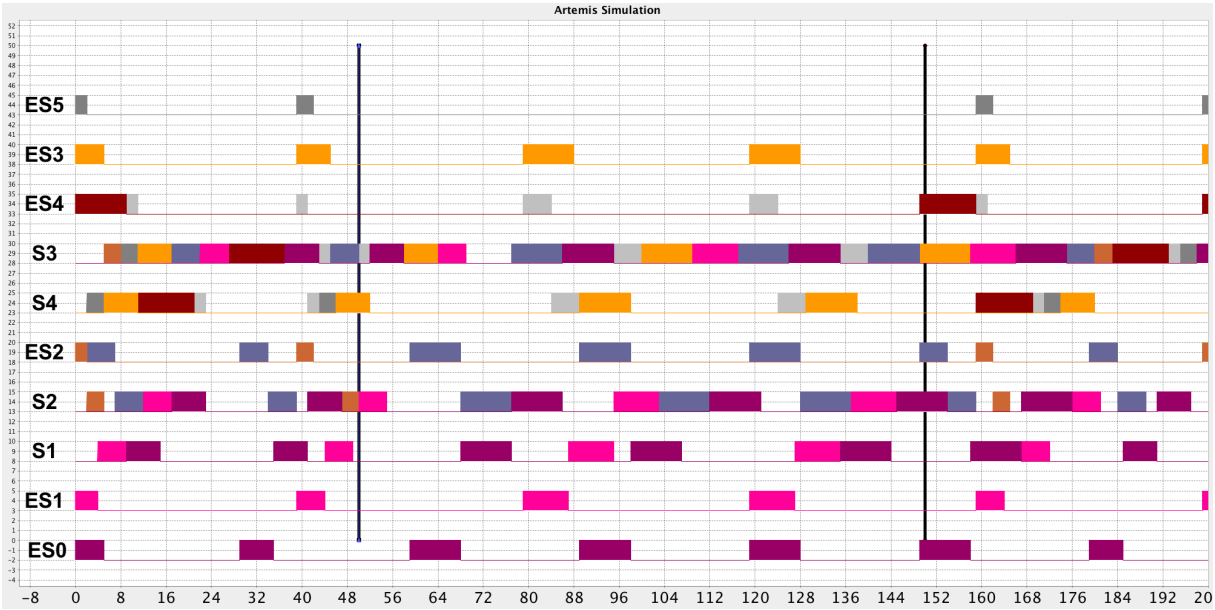


Figure 10.15: Static centralized simulation - Uniform model

We observe that non critical flows (v_3, v_6, v_8) are filtered and their transmission is stopped during the critical phase ($[50; 150]\mu$ s). If we focus on end-systems ES_2, ES_4, ES_5, among the transmitted messages, the LO-critical messages are not transmitted during the interval $[50; 150]$.

The integration of static centralized MC management protocol inside ARTEMIS is effective: the simulated network nodes are correctly reacting the the criticality mode change, conformly to what was

defined during the specifications of the centralized MC management protocol. We also observe that the transmission of the messages is non-preemptive: once a message has been started to be transmitted, it cannot be stopped, even when a criticality mode change occurs (nodes S_3 and S_4 at $t = 50\mu$s).
The centralized static model used shows that there is a high potential number of messages exceeding their LO-WCAT. The probability to have long periods without a critical message is weak (this probability can be expressed as a combination of each uniform law managing each flow). In progressive gaussian model, there is a high probability to stay in HI during the whole time interval. This is confirmed for longest simulation times ($> 1000\mu$s).
We ran the same simulation with anti-progressive transmission time computation model (see figure 10.16). This results shows that the number of potential criticality changes is lower with this model. This is coherent: as computed transmissions times tends to be low, LO-WCAT exceedings are not frequent.

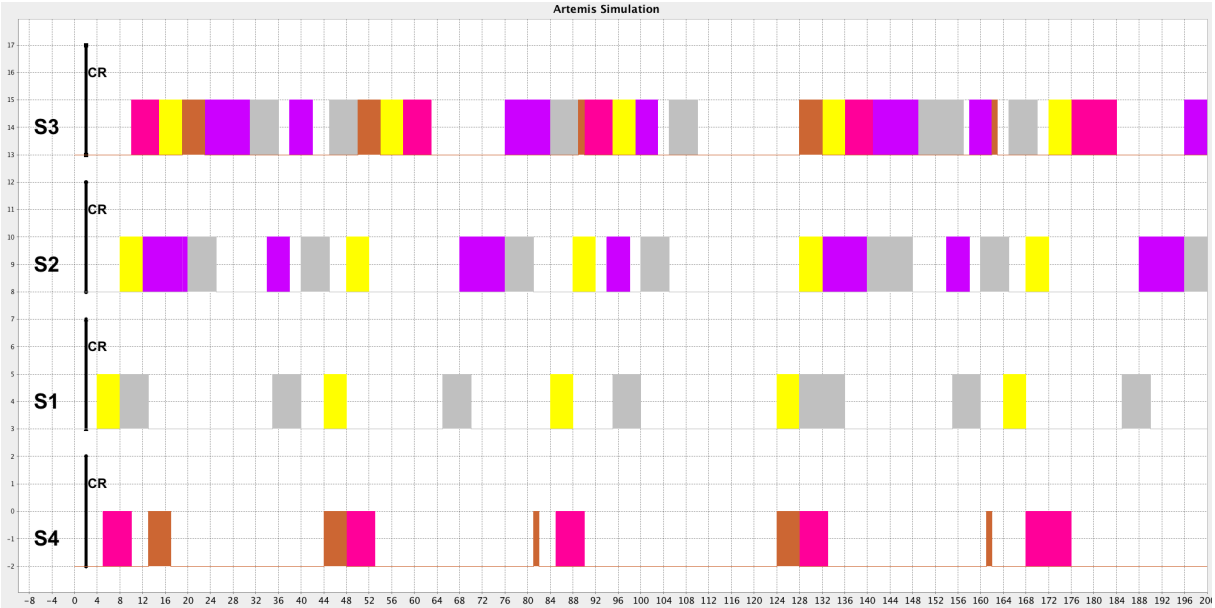


Figure 10.16: Dynamic centralized - Anti progressive gaussian model

We spotted that, every time a message exceeds its LO-WCAT (for example in ES_2 at $t = 60\mu$s), we trigger the change to HI-level. On the contrary, we have to wait a period of 100μs without exceeding before going back to LO level (2 times the highest period). There is no such period in this simulation. This property can be observed in longest simulations ($> 1000\mu$s) where we observe criticality changes back to LO level.

Dynamic example

In order to illustrate the potential of ARTEMIS to trigger criticality level changes by itself and to verify the reliability of the transmission time computation model, we managed simulations in dynamic MC management modes. We kept the network parameters previously defined (topology and flows).
Figure 10.18 and figure 10.17 shows the impact of progressive gaussian and linear models on transmission time computation. Both these models tend to generate high values of transmission times, which is confirmed by the simulation: there a criticality switch to HI mode directly at the beginning of the sim-

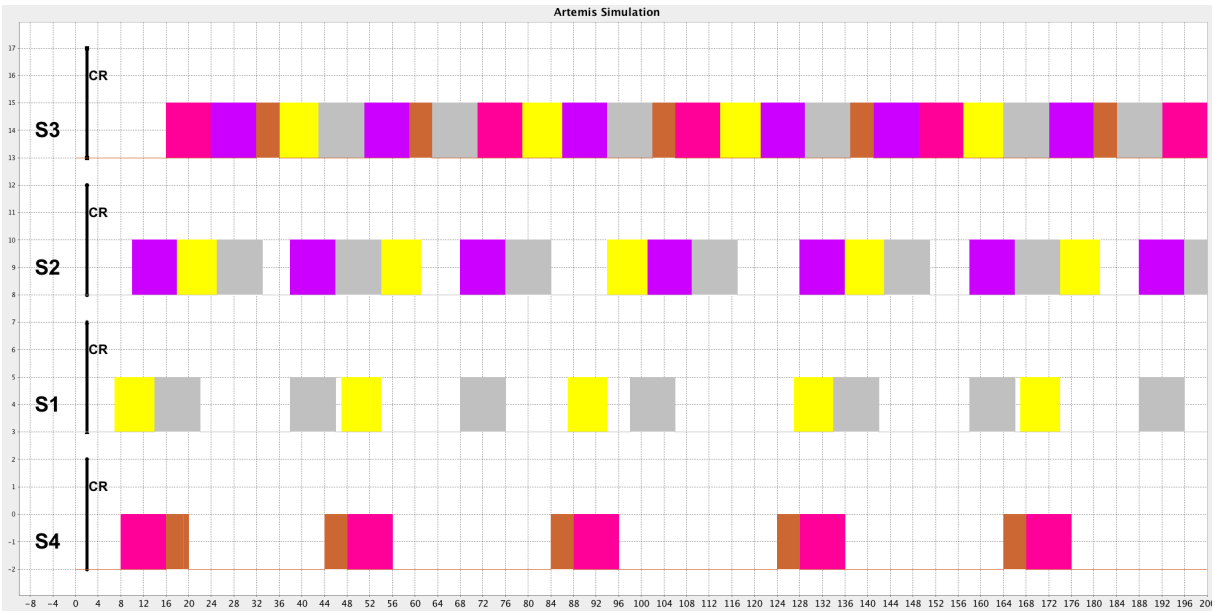


Figure 10.17: Dynamic centralized - Progressive gaussian model

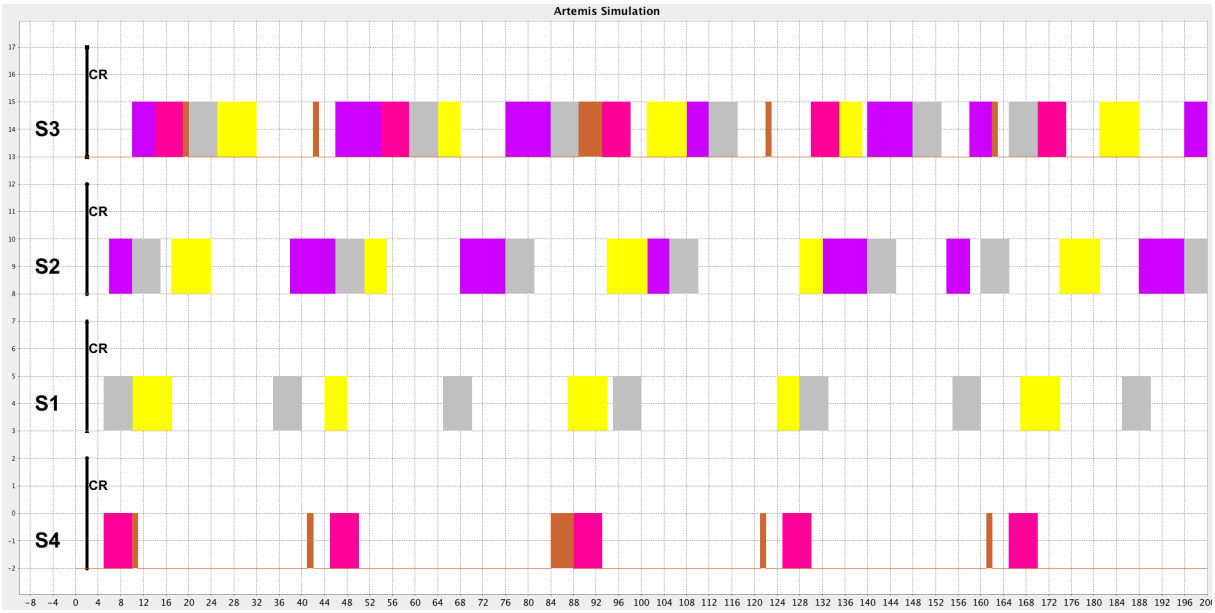


Figure 10.18: Dynamic centralized - Uniform model

ulation, and all the simulation runs in HI mode. That is due to the fact there there is no sufficient idle delay during which there is no HI message transmission in the network. As a conclusion, the network stays in HI mode during the whole simulation.

This shows that the rate of the progressive and linear models has to be precisely selected. If this rate is too high (> 0.8) or too low (< 0.2), generated transmission times tend to be largely below the needed limits to trigger criticality changes.

10.5.2 Decentralized approach

In order to focus on the impact of the decentralized approach, we ran another set of simulations (with linear and gaussian models), but based on the decentralized management of MC. We obtain the figure 10.19.

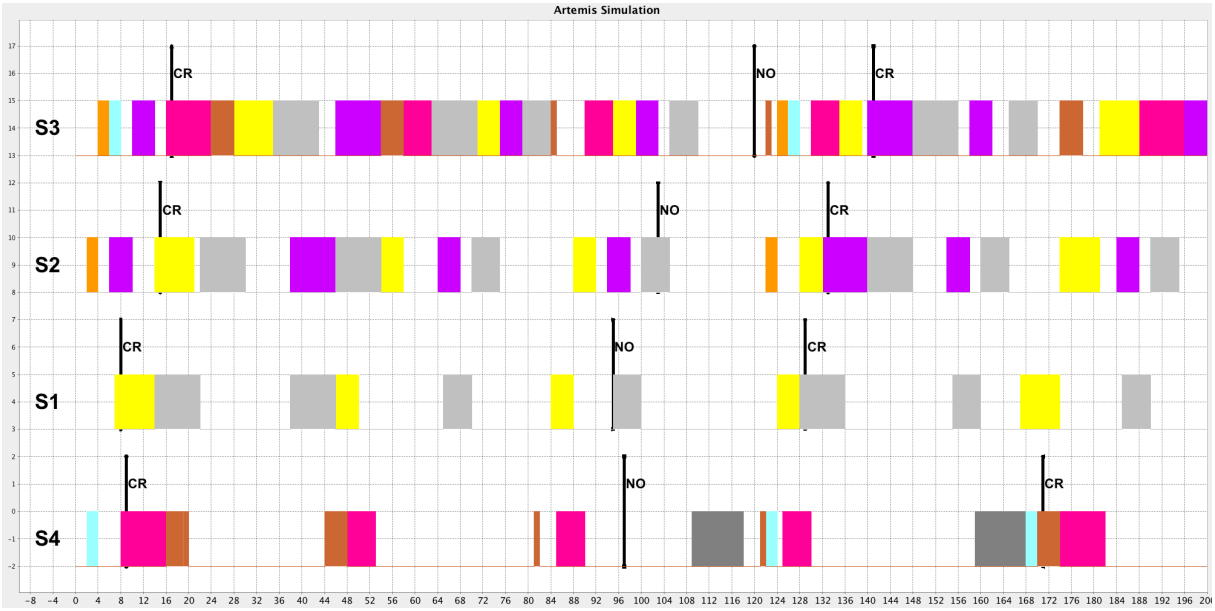


Figure 10.19: Dynamic decentralized - Uniform model

This figure shows that, at the reception of message from v_5 (in pink), the different criticality levels of switches S_4 and S_3 is increased to CR (HI-critical). In S_3, this decentralization of criticality managements allows messages from v_3 (violet) to be correctly transmitted in the switch before changing the internal criticality level of S_3 at $t = 17\mu$s. The same way, both S_4 and S_3 switches are changing back to LO modes at different instants $t = 97\mu$s for S_4 and $t = 120\mu$s for S_3). During this interval, LO-critical flows from v_5 (pink) and flow v_6 (darkgrey) can be correctly transmitted even in LO-mode, according to this decentralized MC management.
Anti-progressive gaussian-based simulation with the decentralized approach (figure 10.20) shows that the end-to-end transmission delays of messages tend to be lower with this model. As a conclusion, potential criticality level changes are triggered later in the simulation. It is due to the point that there is a lower number of messages likely to trigger a criticality increase. A longer simulation interval shows that, in this mode, the time spent in LO-mode is longer.
The progressive-gaussian based simulation (figure 10.21) tends to show opposite results, leading to the same conclusion. With progressive-gaussian model, transmission times tend to be higher than average. That increases the probability to trigger criticality level changes.

10.6 Conclusion

The obtained results shows that the integration of MC management models in ARTEMIS core cover the different MC integration solutions provided in part III. The transmission time computation models and

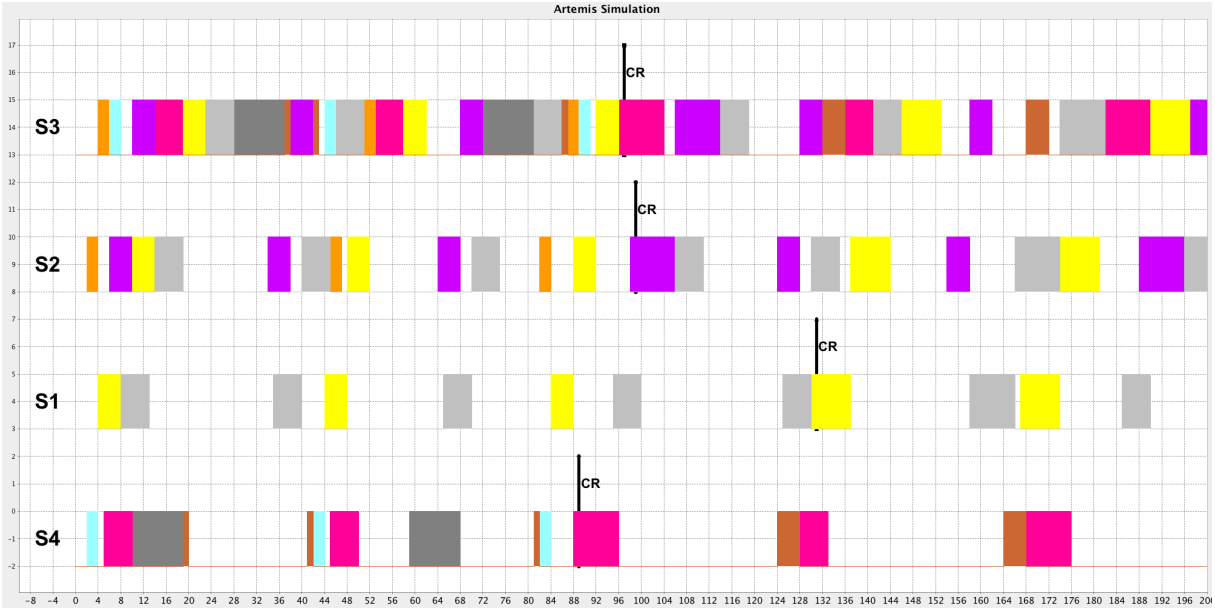


Figure 10.20: Dynamic decentralized - Anti-progressive gaussian model

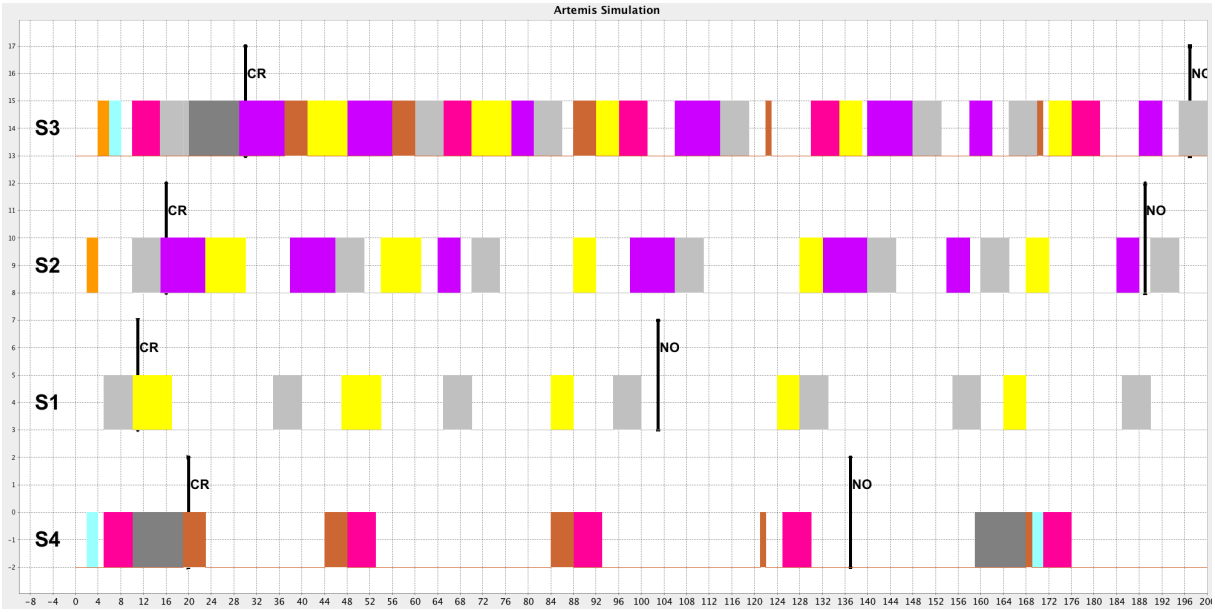


Figure 10.21: Dynamic decentralized - Gaussian model

the different degrees of control the user can adopt to define network simulations through ARTEMIS allows him to represent a various set of network configurations.
From the point of view of what has been shown in this chapter, ARTEMIS can be considered as a reliable MC integration simulation. Combined with the modellings solutions previously presented, ARTEMIS can be used as a solution to validate the dimensioning of an embedded network integrating MC constraints. The provided timing and schedulability analysis can be used as a fundamental in industrial and commercial implementations of RT networks.

Chapter 11

Mixed criticality modelling in simulation tools

"Trop de santé mentale, c'est peut-être ça la folie. Et le plus fou de tout serait de voir la vie comme elle est, et non comme elle devrait être !"

"Too much sanity may be madness — and maddest of all: to see life as it is, and not as it should be!"

– Miguel de Cervantes [136]

Contents

11.1 Introduction

In industrial and research contexts, we need to be able to run successive simulations based on pre-established parameters. For example, establishing a reliable timing analysis for the performances of the decentralized MC management approach can imply to run, at least 1000 to 10000 different simulations, depending on different network topologies and flowsets. It is obvious to mention that all the parameters for all these simulations cannot be defined by the user in a reasonable time. That is why we need to define specific modules to automatically generate topologies.
In order to answer to this need, we defined two different generation algorithms:

- A topology generator, whose role is to generate a given set of a specific number of interconnected nodes.
- A traffic generator, whose role is to generate random flows and to link them with a network topology.

In this chapter, we propose to detail the functional and algorithmic choices that were made in order to build these generation modules.

11.2 Topology generator

11.2.1 Density rate

Generating a topology consists in building a link architecture in order to interconnect a set of nodes. As a result, the first parameter we need to define as an input in our generator is the number of end-systems we target. Inside the topology generator, we consider an end-system as an abstract flow modeller and message sender. We do not apply any physical or electronical constraint to the generic definition of and end-system we provide: an end-system is, basically, only composed of a name, an identifier and a network address. The virtualization layer provided by the Java virtual machine creates a clear separation between the physical and the simulation layer.
When we want to generate a topology, we specify the number of end-systems we target. It automatically builds a set of end-systems and associates a unique identifier to it. In order to create a topology, we need to define the switches and the links between all the created end-systems and switches.
To do this, we adopted a creation method based on a uniform repartition law. Each node has a predefined probability to be linked to another one. This probability is based on a parameter defined by the user. This parameter is the density rate α of the topology. This is a ratio $0 < \alpha < 1$ which defines the probability for two consecutive nodes to be directly linked. If we take two consecutive end-systems in the set, they have a probability α to be directly connected to the same switch.
This set of end-systems is sorted by the identifiers of the end-systems it contains. Each end-system, when created, is associated to an identifier included in $[1; n]$. This is the identifier we use when building the links. If we suppose a generated topology of n_{ES} end-systems and two end-systems ES_i and ES_j, we can define the probability for them to be directly linked to the same switch as:

$$\mathcal{P}_{link}(i,j) = \alpha^{j-i} \tag{11.1}$$

11.2.2 Generation algorithm

The generation of switches is based on a recursive approach. Once we applied the described method on all end-systems, we obtain a first set of switches. One this has been done, the first created set of switches has to be considered as a set of points to connect, to which we apply again our method. We consider these created switches as a set of nodes to connect. Two consecutive switch will be directly connected to a third one or connected to two different switches, depending on the value of α. The generation algorithm is detailed in algorithm 5.

```
Data: Number of end systems n_s,
      connection rate α
Result: Network topology 𝒩
1  j ← 0
2  for i ← 0 to n do
3      r ← random(0, 1)
4      if (r ≥ α || i == 0) then
5          S_j ← new Switch
6          𝒩 ← 𝒩 ∪ S
7          j ← j + 1
8      end
9      link(ES_i, S)
10 end
11 i ← 0
12 while i < j do
13     r ← 𝒰(0, 1)
14     if (r ≥ α || i == 0) then
15         S_j ← new Switch
16         𝒩 ← 𝒩 ∪ S
17         j ← j + 1
18     end
19     link(S_i, S_j)
20 end
```

Algorithm 5: Topology generation algorithm

Each node has a probability of α to share a switch with the previous end system in the list, and a probability of $1-\alpha$ to be linked with a new switch. The loop is based on two indexes, i and j. i represents the number of nodes which have already been connected to the network, as j represents the total number of generated nodes. Once we verify the condition $i = j$, it means that the generation is terminated. At each loop, we generate a random value r, according to a uniform random generation low. Then, we verify the condition $r > \alpha$, we generate at least one additional node. It means that the generation process terminates when all the unconnected nodes are attached to the same new one. Basically, the value of r is computed according to a uniform law $\mathcal{U}$. But, in a future work, we can suppose alternative distribution laws to compute r.
Once the algorithm is terminated, we can guarantee that each node is linked, to at least one other node. One node can has one or zero node as output. The number of inputs is not limited.

11.2.3 Performances tests

In order to evaluate the impact of α on the generation time, we wanted to test the topology generation algorithm with different values of α. The purpose was to observe the evolution of the generation time of a topology depending on the size of the generated network. Our model tends to show that the lower

α, the higher the number of switches and, as a conclusion, the bigger the network. We implemented a performance test module to configure and launch successive simulations with the topology generator.
We operated different sets of simulations with topology generator. The results are shown in figure 11.2. In order to cover wide network architectures contexts (such as AFDX where there is more more than 200 end-systems), we ran simulations up to 300 end-systems as inputs. In order to get coherent and dense networks, α values were made evoluting between 0.3 and 0.8. Each value of the obtained results corresponds to 30 successive generations.
The results shows (see figure 11.2) that, as expected, the generation time (in ms) tends to increase with the number of end-systems. The more end-systems, the more loops the algorithm needs to do to connect all the nodes. Following the same process, the lowest α, the higher the number of generated switches. As a conclusion, the generation time increases as α tends towards 0.
We also observe that the increasing of the generation time is not linear. Below 120 end-systems, the generation time stays manageable. But, starting around 150 end-systems, the curve is not anymore linear, and the growth curve increases, representing longer generation processes. This can cover simulation purposes, but can appear to be too low when representing very wide topologies (specific avionic configurations can increase up to 400 end-systems).

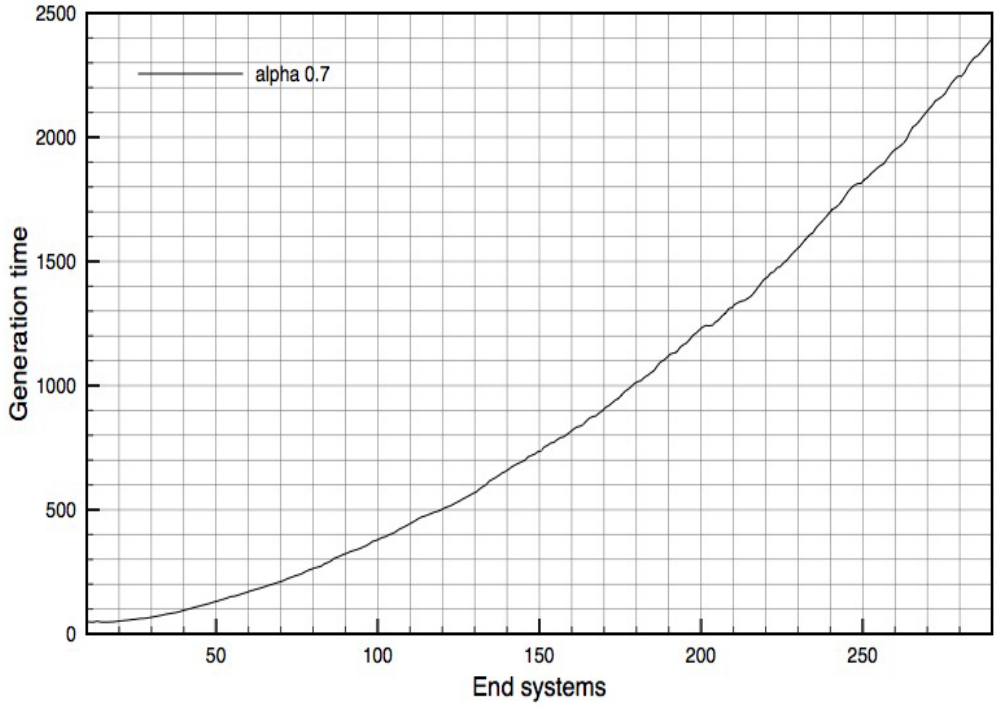


Figure 11.1: Topology generator performances tests with $\alpha = 0.7$

We observe that when $\alpha \leq 0.6$, there is a gap in the generation time. It means that, for high values of the density rate, it has to be taken into account that the performances of the topology generator quickly decreases. The generation delays are satisfying when α $[0.65; 0.85]$.

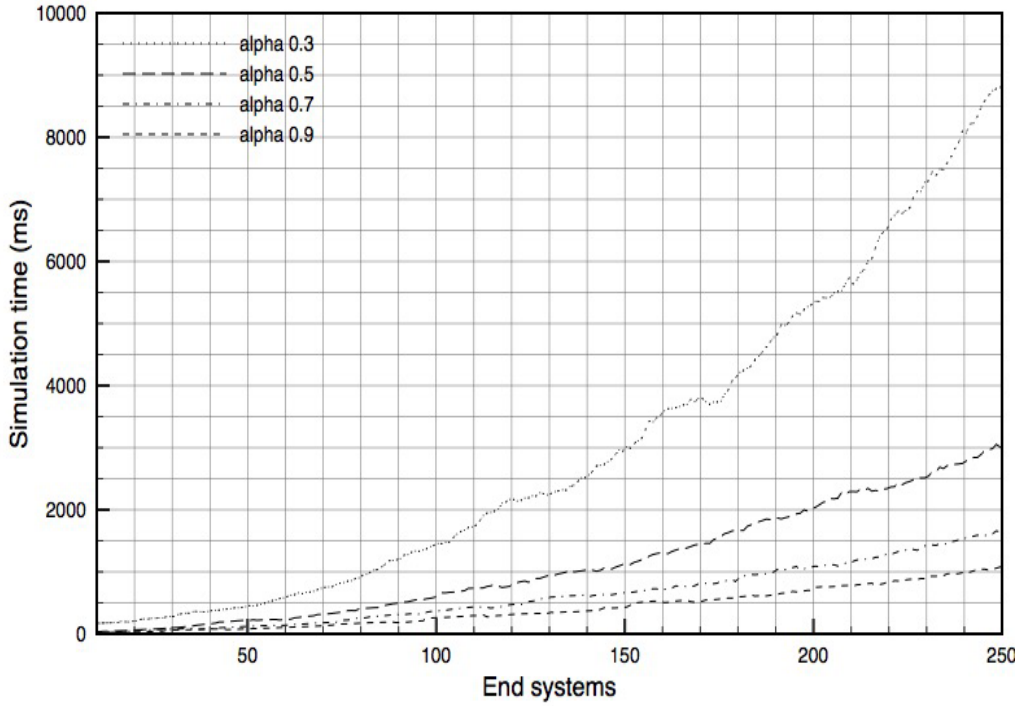


Figure 11.2: Topology generator performances tests

11.3 Flowset generator

UUnifast algorithm is a taskset generation algorithm, based on two elements: a number n_m of tasks to generate, and a target load l. In order to generate flowsets for network simulation, our first intuition was to reproduce UUnifast structure in a network oriented algorithm. This quickly led us to a problem: the modelisation of the load in a set of processors is not simimar to the load modelling in a network topology.

The global load, in processor context, is the combination of all the utilization of the generated task. This load also represents the global load the system has to schedule during the simulation. In network context, this is different. Each flow has a dedicated path, which means that the individual utilization of the flow will have an impact on specific nodes. As a conclusion, in network context, there is a difference between the global load, represented by the cumulative utilizations of each generated flow, and the individual load of each node, represented by the cumulative utilization of each flow transiting through this node. Adapting UUnifast to network context implies to take this constraint into account.

The flowset generation process is splitted in three steps: First, we define a generation algorithm of flowsets similar to UUnifast, in order to generate a global set of flows for the whole network. Then, we build a path attribution algorithm in order to share the load homogeneously among all the network topology. Finally, we show how to integrate MC in this algorithm. We detailed these different steps in the following work.

11.3.1 UUnifast-based generation

In our work, we consider that the WCET of a task is represented by the WCAT of a flow. This is the same for the period parameter. Generating a task or generating a flow can be considered as the same process: generating a couple $\{C_i, T_i\}$ for each flow v_i (or for each task τ_i).
UUnifast generation process is based on two input parameters: the size n of the taskset and its targetted load l. The generation process is as follows: first, we generate the period of each task according to a uniform law, and then we compute the utilization of each task. Then, we obtain the WCET of each task and we control the final computed load: if it is compliant to the target load, we keep the taskset. Otherwise, we restart the generation.
In order to build a flowset generation algorithm, we adopted the same approach as for taskset generation. We built the generation algorithm detailed in algorithm 6. This algorithm is based on two parameters: the flowset size n_m and the targetted global load l.
This value l is the global load represented by all the generated flows. This load has not the same meaning as in processor context: it is only representative of the global traffic, but not of the individual load locally computed in each node. Even if this load has no concrete representation (it is not representative of the load of the network), we keep it in the algorithm in order to be compliant to UUnifast method.
Contrary to processor context, defining $l > 1$ does not necessarily induce an overload in the network traffic. All flows will be spread among the nodes according to their paths, and the global load of the flowset will be dispatched all over the network. The repartition of the load among the nodes depends on the path attribution algorithm (see algorithm 9). The generation process is very similar to classical UUnifast.

Data: Number of messages n, load l
Result: Flow Set $\mathcal{V}$
$\mathcal{V} = \emptyset$
for i *from* 1 *to* n do
  $r_i = \mathcal{U}(log(T_{min}), log(T_{max} + T_g))$;
  $T_i = \lfloor \frac{e^{T_i}}{T_g} \rfloor * T_g$;
  if $i == n$ then
    $u_i = l - \sum_{i=1}^{n-1} (u_i)$;
    if $u_i < 0$ then
      $\mathcal{V} = \emptyset$;
      discard();
    end
  else
    $u_i = \mathcal{U}(0, 1)$;
  end
  $C_i = T_i * u_i$ $v_i = \{T_i, C_i\}$;
  $\mathcal{V} = \mathcal{V} + v_i$;
end

Algorithm 6: Flowset generation algorithm

For each flow v_i in the network $\mathcal{N}$, we generate a random value r_i based on a uniform law $\mathcal{U}$. This value r_i will be used as a base to compute the period T_i of the flow. This method guarantees an homogeneous generation of periods, getting periods of various durations. All period durations will be included between T_{min} and T_{max}.
Once we got the period T_i of a flow, we use a uniform law to generate its utilization u_i. Except for the last flow v_n, each value of u_i is generated to a uniform law $\mathcal{U}(0, 1)$.
At the end, when generating the flow v_n, we compute its utilization u_n as the difference between the target load l and the cumulative utilization of all already generated flows v_1, v_{n-2}, v_{n-1}. If we obtain an negative utilization $u_n < 0$, it means that the generated load already went beyond the target load l. In that case, we discard the taskset. In the other case, the flowset is considered as validated.

Bounding the values

The flow generator has been designed to be used first in Ethernet simulation. As a result, the generated flows must respect the Ethernet standards, especially in terms of size. Each generated message size has to be integrated between 64 and 1518 bytes. These constraints have to be applied to each generated flow. A naive approach would consist in integrating these constraints directly when executing the generation algorithm presented in 6. But that would present two problems. First, a loss of genericity in the algorithm itself by designing it for Ethernet purposes. Then, modifying WCAT during generation could impact the performances of the generation module, particularly in terms of number of discarded flowsets.
As an answer to these problems, we added an additional layer in the generation module, responsible of applying network standards. The algorithm responsible of bounding the message size is described in 7.

```
Data: Flow Set 𝒱
Result: Flow Set constrained 𝒱
for i in 𝒱 do
    if C_i > C_sup or C_i < C_inf then
        α ← random(C_inf/C_i, C_sup/C_i)
        C_i ← C_i * α
        T_i ← T_i * α
    end
end
```

Algorithm 7: Ethernet bounds integration algorithm

For each flow v_i, we check if its WCAT C_i is between the lower and upper bounds C_{inf} and C_{sup}. For classical Ethernet 100 Mb/s, we have $C_{inf} = 5,1\mu$s and $C_{sup} = 121\mu$s.
If $C_i < C_{inf}$ or $C_i > C_{sup}$, we generate a random value alpha (according to a normal distribution law) included between $\frac{C_{inf}}{C_i}$ and $\frac{C_{inf}}{C_i}$. Once this has been done, we compute $C_i = \alpha * C_i$ and $T_i = \alpha * T_i$. The ratio $\frac{C_i}{T_i}$ is not modified by these new values, which means that the individual utilisation represented by the flow is not impacted by this modification.

This algorithm assures that the generated WCAT conform to Ethernet standards in terms of size. The figure 11.3 shows the result of 100 successive generations of 80 flows, in a system composed of 5 criticality levels. These levels are as follows:

- Non-Critical (NONC), with a criticality rate of 0.0 (all flows belong to non-critical level).
- Critical (CRIT), with a criticality rate of 0.2 (80% of flows are critical).
- Mission-Critical (MISS), with a criticality rate of 0.3 (70 % of flows are mission-critical).
- Vehicle-Critical (VEHI), with a criticality rate of 0.5 (50 % of flows are vehicle-critical).
- Safety-Critical (SAFE), with a criticality rate of 0.8 (20 % of flows are safety-critical).

Figure 11.3 shows two results: first, each generated WCAT, for each criticality level of each flow, conforms to Ethernet standards in terms of minimum and maximum number of bytes. Second, the average WCAT of flows (for a specific criticality level) tend to increase as the criticality level is determined as more critical.

Discarding rate

UUnifast is a taskset generation algorithm. It is based on a discarding logic. It means that if the taskset is valid from the point of view of the load, we validate it, otherwise we discard the taskset and we generate

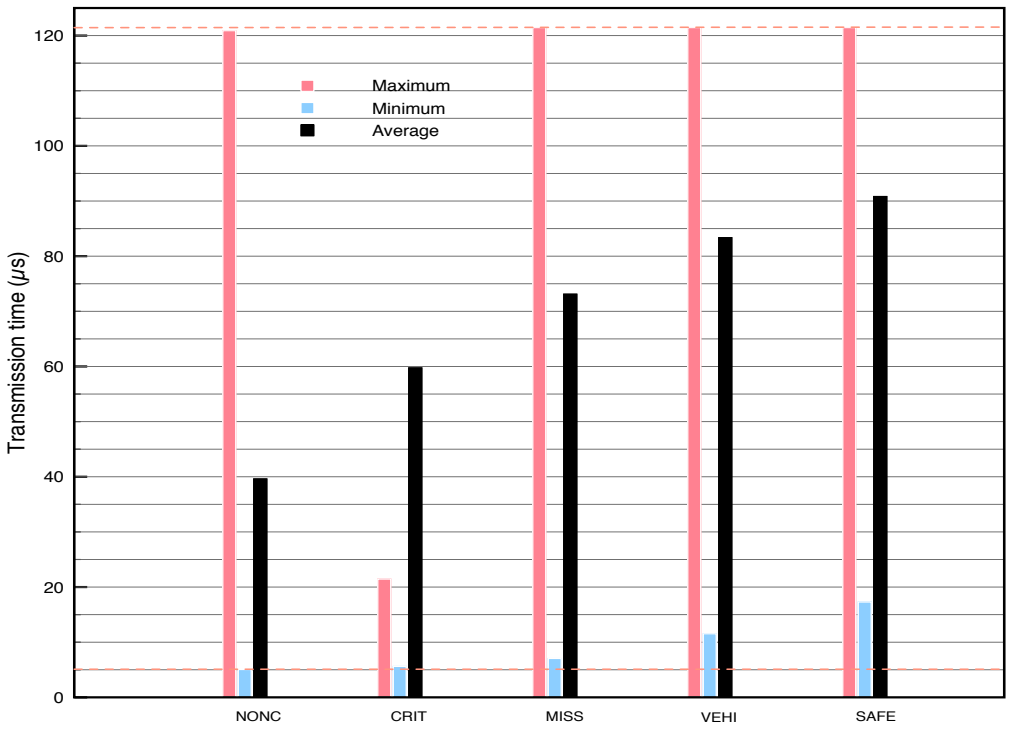


Figure 11.3: Ethernet bounds applied on generated flows

another one. If we focus on the algorithm 6, we observe that if the generated flow has a load which is not equal to the target load, we completely discard the flow set.

The problem of this method is that it is consuming time: generating a totally new taskset implies resetting the generation process from the beginning. This represents a loss of time and ressources. It can be ignored when generating just a few tasksets (for example purposes). But it has to be taken into account when we need to generate large amounts of tasksets (100.000 - 1.000.000), which happens frequently during simulations.

When it comes generating flowsets, we face exactly the same problem. In order to evaluate the loss of data and time represented by these discards, we generated flowsets automatically for given values of targetted loads. To do this, we define the acceptance ratio of a generation. This correspond to the ratio of accepted flowsets compared to all the flowsets which were generated by the algorithm. We obtained the results described in figure 11.5.

UUnifast provides an acceptance ratio which does not depend on the number of generated flows. Even for a high number ($n \geq 150$), the acceptance ratio stays around 17 %. This simulation (see figure 11.5) was done for a targetted global load of $l = 0.5$. Each value of n corresponds to 100 different flowsets. As a conclusion, UUnifast algorithm provides a solution to generate wide flowsets without increasing the generation time as the number of flows increases.

We operated different simulations, corresponding to various load values (from 0.5 to 0.8). These simulations tend to show that the acceptance ratio of the flowsets depends on the load. To confirm this intuition, we ran successive generations of flowsets with $n = 50$, for different values of l (from 0.2 to 5.0). The results (see figure 11.5) shows that the acceptance ratio of the algorithm decreases with high

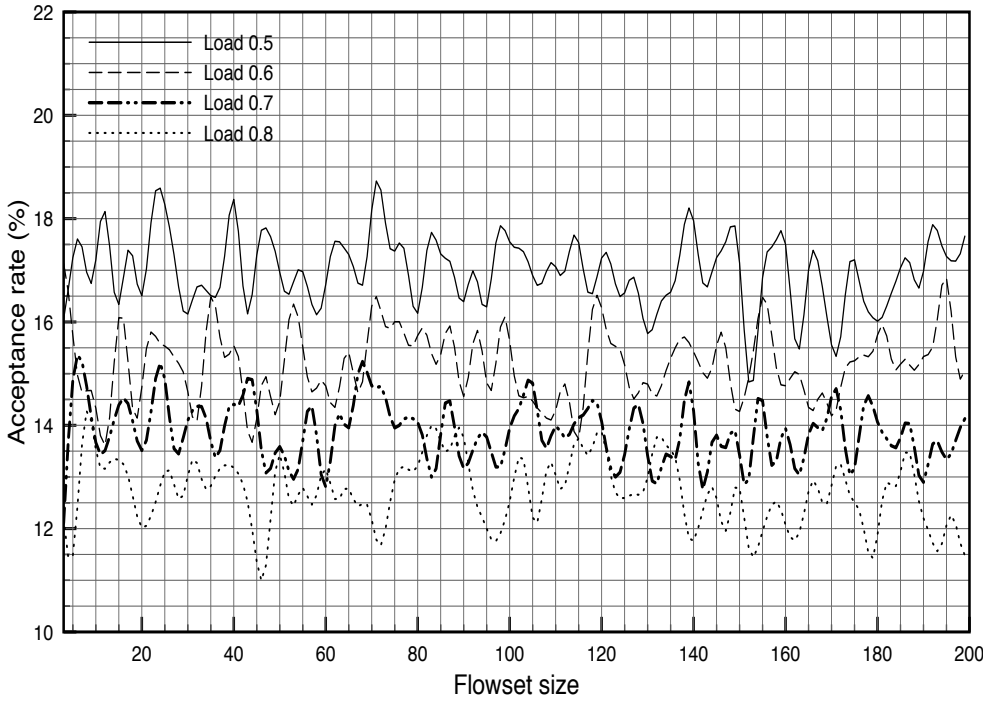


Figure 11.4: Discarded flowsets / Flowset size

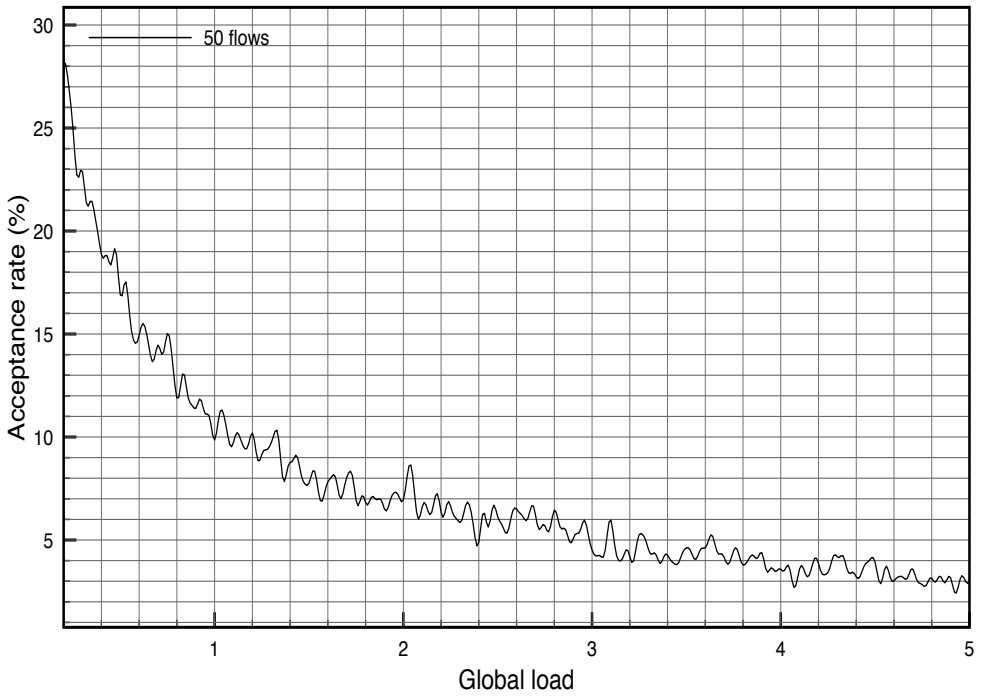


Figure 11.5: Discarded flowsets / Load

values of l.

As a conclusion, the higher l, the lower the acceptance ratio. In order to answer to this, we want to propose modifications to UUnifast. These modifications, detailed below, allows us to decrease the number of discarded flowsets per flowset generation process.

Utilization generation

In UUnifast, there is only one reason to discard a generated flowset: the generated flows led to a global load exceeding the value of the target l. In the algorithm 6, this situation is represented by getting a flow of negative utilization.

Uunifast generation process is based on the generation of different utilizations $\{u_1, ..., u_{n-1}, u_n\}$ for all the n flows of the network $\mathcal{N}$. The computation of each value of u_i is, basically, indexed on a uniform distribution law. In order to increase the acceptance ratio of our algorithm, we propose to modify this model by replacing it with a gaussian distribution.

Generating an utilization with a uniform law between 0 and 1 could imply to generate flows with high utilization, representing most of the load of the flowset. The solution we propose is to assure an average utilization for each flow, to anticipate the global utilization of the flowset. The average utilization for each flow is designed to be the same, equal to a balanced repartition of l among all flows. As a conclusion, we propose to build a gaussian distribution $\mathcal{G}$ each flow utilization centralized around the mean value $\mathcal{G}_{mean} = \frac{l}{n}$.

Defining a gaussian distribution law implies defining its variance. Each utilization of each flow has to be bounded: 0 for the lower bound, and l for the upper bound. The variance of the law impacts the randomness and the homogeneity of the flowset. The higher the variance, the more we can find different values of u_i. When we built the algorithm, we computed the variance effect for different flowsets and configuration parameters in order to compute variance values making the generated flowsets pertinent without making the generation time explode.

We need to implement a variance that verifies: $0 < u_i < l$. Given that each value of u_i is centered around $\frac{l}{n}$, the variance of the distribution law will be computed as the smallest interval between $\frac{l}{n}$ and the different bounds. As a conclusion, the variance of the law is computed as follows: $\mathcal{G}_{var} = min(\frac{l}{n}, l - \frac{l}{n})$.

We ran classical uniform-based UUnifast and gaussian-based algorithms during a set of successive simulations. The results are shown in figure 11.6. In order to compare the acceptance ratio of both algorithms, we ran simulations of flowsets of 50 flows. For each value of the global load (from 0.2 to 2.0), we ran 100 different simulations.

The simulation results shows an increase of the acceptance ratio with the gaussian model. Both models tend to decrease the acceptance ratio with the increasing global load, which was already observed before. In our model, there is a difference of around 5 % between the models. The variance has a strong impact of the simulation results, and on the acceptance ratio. The lower the variance, the higher the acceptance ratio. In that simulation case, the computed variance is varying between $0,004$ and $0,04$.

As a conclusion, the gaussian model we propose to integrate, as an alternative to classical UUnifast, provides a higher acceptance ratio: the gaussian-based algoritm provides less discarded generated flowsets. The uniform-based algorithm provides more heterogeneous results as the gaussian-based algorithm provides a solution which is more performant, as requiring to discard less generated flowsets. It is up to the user to choose between these models.

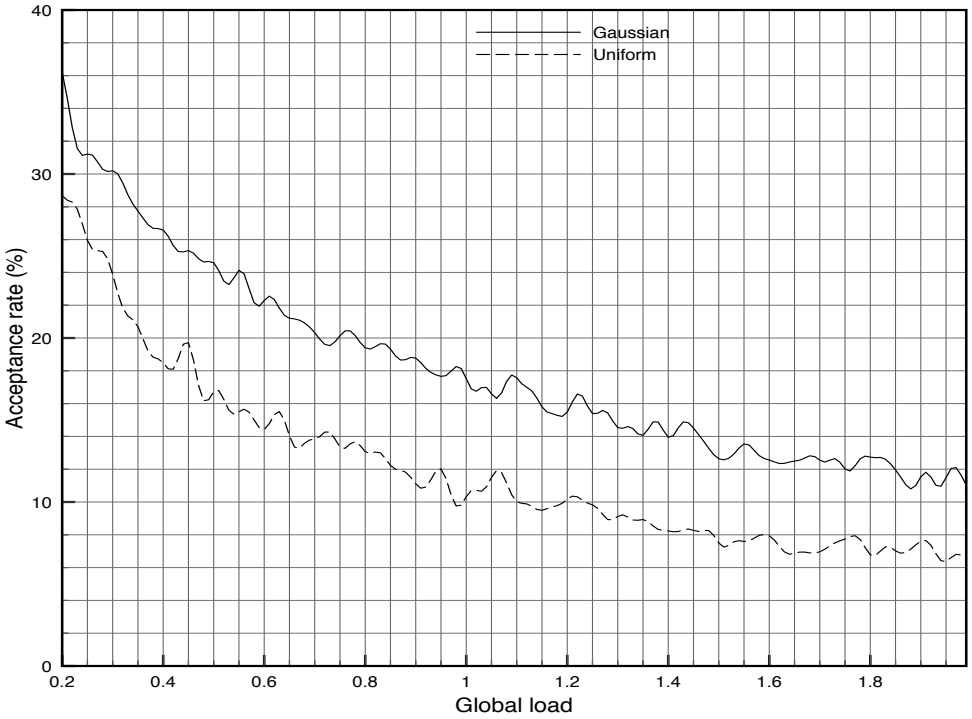


Figure 11.6: Gaussian and Linear generation models results

Massive generation

The gaussian model we proposed represents a strict increase in terms of acceptance ratio in the flowset generation. When requiring to generate high amounts of flowsets of different loads (for simulation and benchmark purposes, for example), the global acceptance ratio of the algorithm can still be improved. Our approach is based on the following assumption: most of the discarded flowsets are discarded due to an overload. Their global load is higher than the target l.

We suppose that we want to generate a wide set of flowsets (from 100 to 100000), each flowset associated with a specific target load l. We want, at the end, to generate a set of flowsets for each value of l between l_{min} and l_{max}. Basically, we propose to generate these flowsets with UUnifast. The total utilization of a flowset is denoted as U.

When the flowset is generated, there is two possibilities: either $U = l$ and the flowset is valid, or $U \neq l$. In that case, instead of discarding the flowset, we propose to keep it as valid for another target load l'. In other words, the generated flowset may still be compliant to another targetted load, for the same set of generated tasksets.

We described the generating process integrating this solution in the algorithm below (see algorithm 8). We suppose that we target to have a set of k tasksets, each taskset belonging to a targetted load $\mathcal{L}_k$. We want to generate, at least, n_t tasksets for each value of $\mathcal{L}_k$, with $\mathcal{L}_{min} \leq \mathcal{L}_k \leq \mathcal{L}_{max}$.

In order to illustrate this process, we ran a simulation to generate various flowsets from 20 to 50 flows. We tested this environment with a evoluting load from 0.2 to 1.0 (100 flowsets per load value). We obtained the results described in figure 11.7. All these simulations have been run with UUnifast based on a gaussian model.

```
Data: n_t, n, L_min, L_max
L_i ← L_min ;
for L_i ← L_min to L_max do
    while n_i < n_t do
        Taskset T_i = UUnifast(L_k, n);
        if L_i == L_k then
            E(L_k) = E(L_k) + {T_i}
            n_k ← n_k + 1;
        else
            if L_i ≤ L_max && L_i > L_k
             && n_i < n_t then
                E(L_i) = E(L_i) + {T_i}
                n_i ← n_i + 1;
            else
                Discard T_i;
            end
        end
    end
end
```

Algorithm 8: UUnifast-Massive algorithm

We note as $\mathcal{E}(\mathcal{L}_i)$ the set of flowsets generated for targetted global load $\mathcal{L}_i$.
Each time the algorithm generates a flowset $\mathcal{T}_i$, we check its load l_i. If it it is equal to its target load l, we associate it with the set $\mathcal{E}(\mathcal{L}_i)$. If we have $l_i > l$, then we keep $\mathcal{T}_i$ and we associate it to $\mathcal{E}(\mathcal{L}_k)$ with $\mathcal{L}_k = l$.
That allows us to discard a flowset only in the case that its load is not compliant to any potential targetted load. With this solution, we can reduce the number of discarded flowsets. That increases the performances of our generation module by decreasing the generation time.

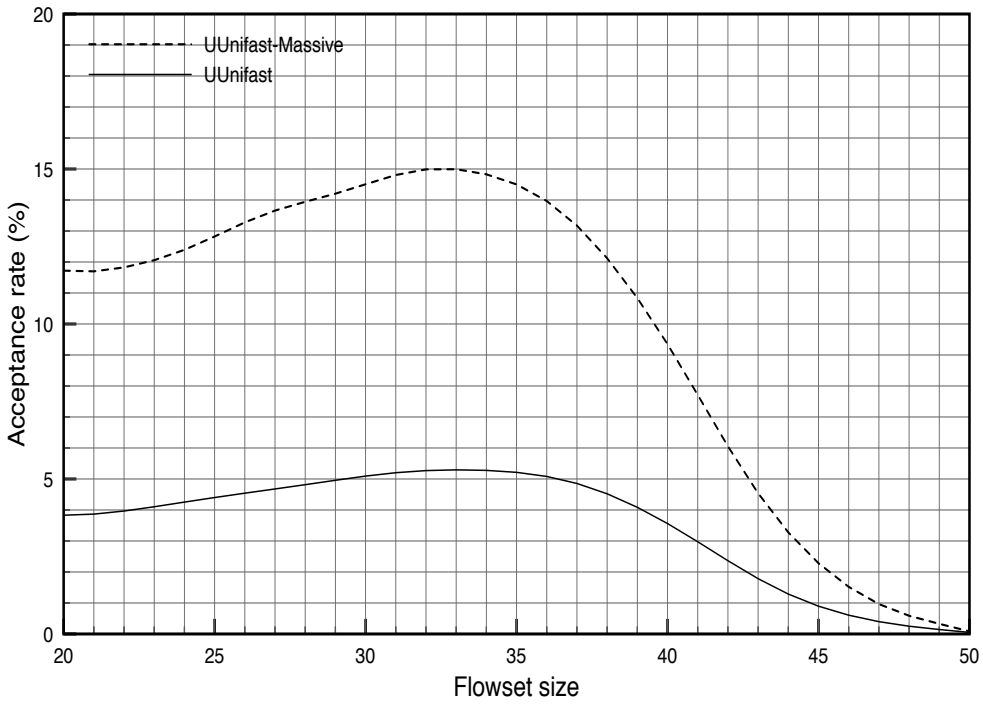


Figure 11.7: UUnifast and UUnifast massive acceptance ratios

We observe a clear difference in the acceptance ratio of both algorithms, particularly between 20 and 40 flows. Beyond this limit, both the acceptance ratios tend towards 0. At the maximum, we observe a difference of 10 % between the two algorithms. This shows that UUnifast-massive solution is a strict

improve of basic UUnifast, in terms of acceptance ratio management.
It means that, among all the flowsets generated by UUnifast, approximately 10 % of them (at least) can be considered as valid for another global load value. We can conclude that the generated flowset is not corrupted nor irrelevant, as its load is still coherent but not adapted to the target load.
We also observe that, as the number of flows increase, both acceptance ratios decrease to 0, which is coherent. The acceptance ratio of UUnifast tends to decrease as the flowset size increases (see section 11.3 for details). Figure 11.7 shows that the acceptance ratio of 30 flows is higher than the one for 20 flows, which goes against what we showed previously. This phenomenon is due to the gaussian model integrated in UUnifast. This model tends to integrate in UUnifast, for each targetted load, an optimal value of flows where the targetted utilization (for each flow) is the easiest to reach. This phenomenon induces a slight increase of the acceptance ratio at low flowset sizes.
As a conclusion, the presented solutions allowed us to build a reliable flowset generation algorithm. The different solutions are extracted from the model of UUnifast presented in the litterature. This represents the fundamentals we based our work on when developing the topology and flowset generators. We showed by successive simulations that it is possible, for our context, to improve the performances of classical UUnifast by decreasing its discarding rate.

11.3.2 Path computation

Generating a flowset starts by generating the period and WCAT of each flow. But, as mentioned previously, this is not enough. We need to attribute a path to each flow in the network. Each flow must have a dedicated path, going from one end-system (as input) to another end-system (as output). That causes a problem in terms of load management: the global load l of a flowset is not sufficient by itself to represent the network load. We need to propose an algorithm to compute the path of each flow and assure a balanced repartition of all the flows among a topology.
In processor context, the load represented by a taskset $\mathcal{T}$ is represented by the expression of $\sum_{i \in \mathcal{T}} (\frac{C_i}{T_i})$. In network context, the expression of the load is different for each switch n in the network $\mathcal{N}$, expressed as $\sum_{n \in \mathcal{N}} (\frac{C_i}{T_i})$. Each switch in the network has its own internal load which can be totally different from the load of its closest neighbours. That leads to the following question: what is an "overloaded" network? To answer to this, we need to define an accurate method to control and adapt the repartition of the load among the network.
Once we generated a flowset with a global load l, we need to associate a path to each flow v_i. In order to manage the individual load of the network and the balanced repartition of flows among a topology, we need to define specific parameters to measure the network traffic. We introduce these parameters as follows:

- The individual load of a node n is denoted as $l(n) = \sum_{n \in \mathcal{P}_\rangle} (\frac{C_i}{T_i})$. This corresponds to the local observation of the traffic transiting through the node n.
- The global load l of the network is represented as the sum of the individual load of each message (and not as the individual load of each node). If we suppose a generated flowset $\mathcal{V}$, we have $l = \sum_{i \in \mathcal{V}} (\frac{C_i}{T_i})$. This is the global load used as input parameters in our flowset generation algorithm.

- The average load $L = \frac{\sum_{n \in \mathcal{N}} (\frac{C_i}{T_i})}{|\mathcal{N}|}$. This parameter is used to observe the global cumulative load of the network $\mathcal{N}$. It represents the average load per node.

We assume that these parameters will allow us to evaluate the network traffic globally.

The UUniNet algorithm

Based on these parameters, we built an path computing algorithm called UUniNet [137]. UUniNet is flowset generation algorithm. Its flow generation process is detailed in section 11.2.2. It integrates a path computation algorithm, taking as input a network $\mathcal{N}$ and a set of flows. That is the part detailed below.
UUniNet path computation is detailed in algorithm 9. It is based on a credit association to each node. Initially, each node n is associated with a dedicated credit w_n. A naive approach would consist on defining w_n as equivalent for each node. But the structure of the network will tend to make the switches of network more important than the nodes. As a result, the credit associated to each switch is higher. In the network $\mathcal{N}$ composed of n nodes, we associate the following credit to each node:

- If the node n is an end-system, we note $w_n = l * \frac{k}{n}$. k is the number of end-systems in the network.
- If the node n is a switch, we note $w_n = l$.

This credit attribution allows us to balance the attribution of paths to the flow. In many classical industrial architectures, the ratio between end-systems and switches can be high (200 for 8 for AFDX, for example). Associating a different credit to the two types of nodes forces the flows to pass through different switches and to create various paths among the network.
At first, we associate a credit to each node by looping to all nodes in network $\mathcal{N}$.
For each flow, we want to assure that its path will start in an end-system. To do this, we pick the first node of each path as the end-system with the highest remaining credit. In the case where two end-systems have the same credit, we randomly pick one of them. We substract from the selected end-system credit the utilization of the current flow.

Simulation results

In order to evaluate the maximum and average load in the network provided by UUniNet, we operated a simulation on various topologies. Each point corresponds to 10 different flowsets generation. Each flowset contains 50 flows.
Simulation results (see figure 11.8) tend to show that, even when the global load l exceeds 1.0, the maximum and average load in the network do not tend to exceed 1.0. There is a gap between maximum and global load. This is coherent: an equality between them will correspond to say that there is a node in the network where all flows transit through (following the hypothesis that a flow cannont transit twice through the same node).
Nevertheless, even it is rare, this situation can happen, for example, in tree-oriented topologies where all flows converge to the same destination. It means that, to assure that there will be no overload in the network, we have to assure that $l \leq 1.0$.

Data: flow set $v_1, ..., v_i$, network topology $\mathcal{N}$
Result: Path attribution for messages from $\mathcal{V}$

```
for each flow n in N do
    if n is a switch then
        n.credit ← l;
    else
        n.credit ← l * k/n;
    end
end
for each flow v_i in N do
    K ← pick_ES_with_highest_credit();
    if K.credit ≥ C_i/T_i then
        K.credit ← K − C_i/T_i
        P_i ← P_i + {K}
    end
    while |P_i| < 3| or last(P_i) is not an ES do
        K ← pick_neighbour_with_highest_credit();
        if K.credit ≥ C_i/T_i then
            K.credit ← K − C_i/T_i
            P_i ← P_i + {K}
        end
    end
end
```

Algorithm 9: UUniNet path computation

At each iteration of the algorithm, we pick the node with the heighest credit among all the closest neighbours of the current node (the last node of the flow path). We add this node to the flow path and we reduce its credit. If this new node is an end-system, the algorithm stops for the current flow, and starts computing the path of the next flow until the end of the process.
If the selected node is not an end-system, the algorithm defines the node as the new current node, and restarts a loop.
For each flow v_i, each time it is associated to a dedicated node n, we substract from w_n the value of the utilization u_i from v_i. We assure that $w_n \geq 0$ for each node, at each time. This induces that the credit of each node represents the maximum load it can endure.

As a conclusion, UUniNet provides an homogeneous repartition of the flows among the network, due to the credit repartition defined in the algorithm. Depending on the topology, we can generate flowsets with global load exceeding 1.0, but this can lead to potential overloads in the network. UUniNet generation process is based on UUnifast and, depending on the model, can provide better acceptance ratios than classical UUnifast, which makes it performant even when generating wide number of flows.
UUniNet flowset generation is functional. We need to improve it by integrating mixed-critical flows generation inside the algorithm. That is the final point of our work on RT simulation: integrating MC in flowset automatic generation.

11.3.3 Mixed criticality integration

When defining flows manually, it is up to the system designer to decide which flow belongs to which criticality levels. But, when generating flows automatically, this decision has to be made based on specific parameters.
Generating mixed critical flows can be splitted with two problematics. First, how do we decide the criticality level of each flow? And, then, how do we compute the WCAT of a flow for each criticality level it belons to? These are the questions we answer in the following part.

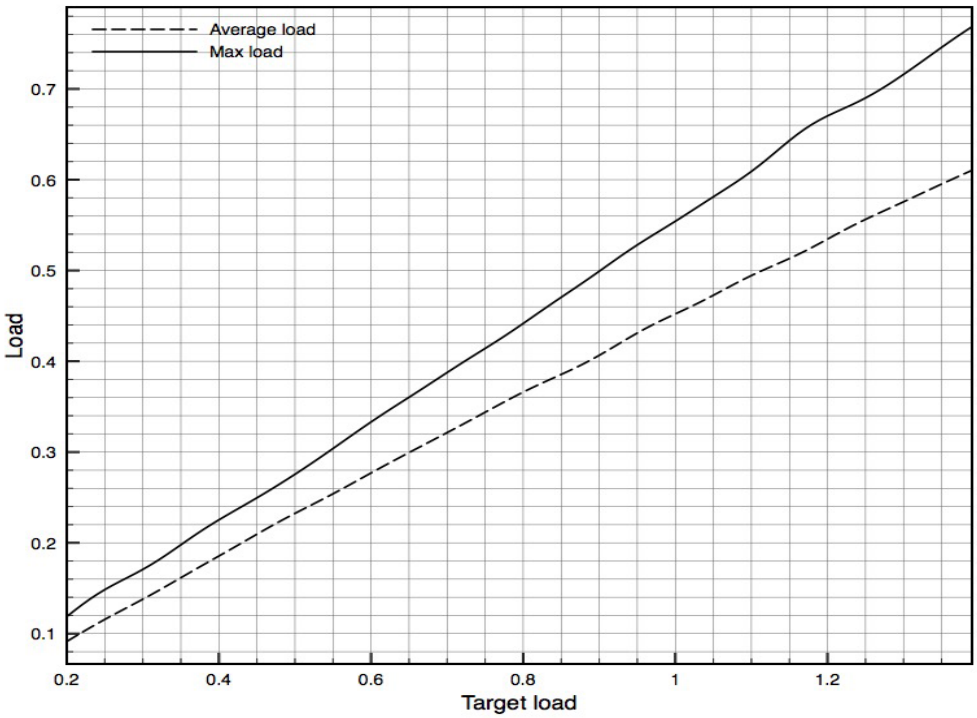


Figure 11.8: Average generated load / target load

Criticality ratio

In order to determine the criticality level of each flow, we define, for each criticality level (except LO a criticality ratio. According to a uniform distribution, this ratio represents the probability (for each flow) to have a WCAT corresponding to this criticality level. For each criticality level $\gamma_1, ..., \gamma_{k-1}, \gamma_k$ of the network, we note α_{γ_k} the corresponding criticality ratio. We have the following assumption: $\forall i \in [1;k], 0 < \alpha_{\gamma_1} < 1$.

In order to respect Vestal's hypothesis [114], the hierarchical structure among all the criticality levels of the network implies that, if a flow belongs to γ_i level, it also belongs to all LO$\gamma_1, ..., \gamma_{i-2}, \gamma_{i-1}$ levels. We have to integrate this constraint in our model. It gives us: $\forall i, j \in [1;k], i < j \implies \alpha_{\gamma_i} > \alpha_{\gamma_j}$. It means that the probability to belong to a criticality level γ_i is necessarily lower than the probability to belong to γ_{i-1}.

In order to implement this model in UUniNet, we generate a unique ratio α^i for each flow v_i. For each α_{γ_j} which verifies $\alpha^i > \alpha_{\gamma_j}$, we generate a WCAT for v_i at the level γ_j.

The solution was to define a unique static ratio r, and to compare each criticality ratio to it. If the criticality ratio r_i is higher than r, then the generated flow belongs to the level γ_i. This hierarchical structure implies that we cannot generate more flows of γ_i level than at γ_{i-1}. At the end, each flow will we characterized by a set of WCAT $C_i^{\gamma_j}$, with $C_i^{\gamma_j} = -1$ if v_i does not belong to the level γ_j.

WCAT matrix

The model we propose allows us to define the different criticality levels of each flow. But there is another problem to solve: how do we determine the WCAT of each criticality level for a flow? First, the hierarchical structure among flows allows us to assure that for any flow v_i: $\forall m, j \in [1;k], i < j \implies C_i^{\gamma_m} \leq C_i^{\gamma_j}$. But we need to define a solution to determine, flow by flow or globally, a solution to compute each value of $C_i^{\gamma_j}$.

One first solution would have been to propose a static constant, acting as a multiplier between two WCAT of the same flow and consecutive criticality levels. But this solution is not representative of real situations and, as a result, we propose a solution based on a random evaluation of the value.

Basically, we consider that the highest WCAT for a flow v_i is equal to its period T_i. This corresponds to an utilization $u_i = 1$. It means that, for each flow v_i, its WCAT for level γ_j will be computed conformly to a linear distribution, between $C_i^{\gamma_{j-1}}$ and T_i (see algorithm 10 for details).

In order to generate flowsets of different criticality levels, we integrate this algorithm inside the flowset generation module (see 11.3).

```
Data: flowset 𝒱, criticality levels
      {γ1, γ2, ..., γk}
Result: Mixed criticality flows 𝒱
for Each flow v_i ∈ 𝒱 do
    α_i = random(0, 1);
    for Each level γ_j ∈ {γ1, γ2, ...γu} do
        if α ≥ α_{γ_i} then
            u_i^{γ_j} = random(u_i^{γ_{j-1}}, 1);
            C_i^{γ_j} = T_i * u_i
        else
            break;
        end
    end
end
```

Algorithm 10: MC integration in UUniNet

For each flow v_i, we generate the value of α_i. It is included between 0 and 1 and, the higher α_i, the higher the maximum criticality level of v_i. For each criticality level of v_i, we generate a dedicated utilization $u_i^{\gamma_{j-1}}$. From this utilization value, we can deduce the value of $C_i^{\gamma_j}$.

This global solution tends to finally generate a matrix of dimension $n * k$, with n the number of flows in the network and k the number of criticality levels. This matrix, combined with the generated path and periods, will represent the global description of all the flows in the network. As a conclusion, UUniNet generates random flows for a given topology, based on a number of flows n and a global load l.

Simulation results

In order to validate the possibility to generate flows with multi-criticality levels, we operated a simulation based on successive simulation. We worked on a context with 5 different levels of criticality: non critical (NONC), critical (CRIT), mission-critical (MISS), vehicle-critical (VEHI), safety-critical (SAFE). We make the global target load change between 0.2 and 1.5, and each point is the result of 100 successive generations. The flowsets we designed are all composed of 50 different flows.

Basically, we defined static ratios between all the criticality levels. The defined ratio were defined as follows:

Criticality Level	NONC	CRIT	MISS	VEHI	SAFE
Ratio (SIM 1)	0.0	0.4	0.7	0.9	0.95
Ratio (SIM 2)	0.0	0.2	0.3	0.5	0.8

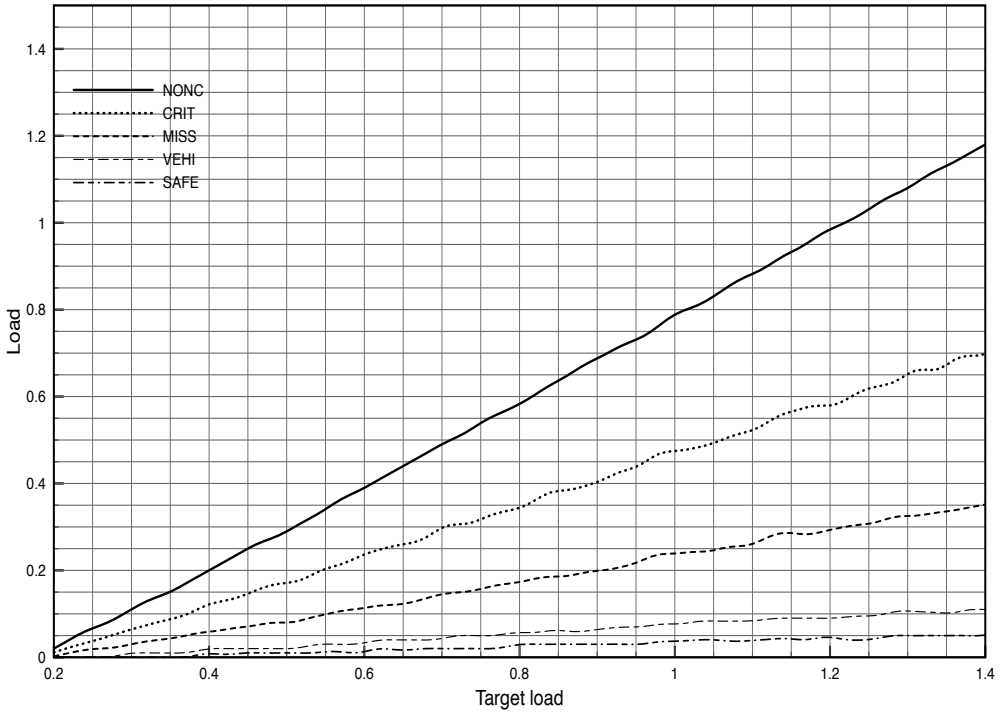


Figure 11.9: Flowset generation with UUniNet (SIM 1)

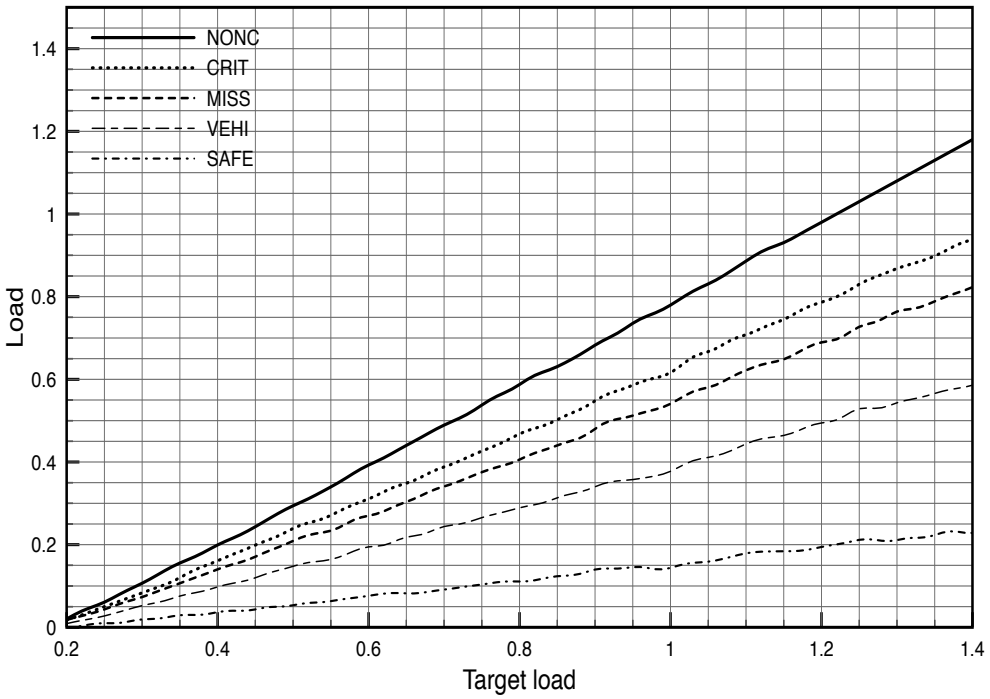


Figure 11.10: Flowset generation with UUniNet (SIM 2)

Both figures 11.9 and 11.10 show that, according to the criticality ratios of each criticality level, the load

represented by the flows tend to decrease as the criticality level increases. Even if higher critical flows tend to have a higher utilization, the reduction of the number of flows is clearly more important.
This reduction of the network load proves that, when increasing the criticality level in the network, we have to assure the transmission or privileged messages, but representing a lower load. As a conclusion, it is easier to transmit.
These simulations shows that our implementation of UUniNet generates flowsets of different criticality levels. The algorithm we designed can be used as a traffic generator for network simulation.

11.4 Conclusion

In this chapter, we presented two generation modules: network and flowset. These generators have been implemented in Java. The model they are based on represents an improve. Being able to generate mixed-critical flows and associate network path to flows are methods which can be reused as standalone projects or as parts of external simulation tools. Thus, the focus we did on improving the acceptance ratio of classical UUnifast allowed us to assure reasonable performances.
To improve this model, a solution could be to propose an algorithm which is able to maintain also a minimum load inside a generated flowset. It will offer additional guarantees on the network traffic induced by the path attribution process.

Chapter 12

Protocols simulation results

"Le monde est bien comique, l'humanité en est la blague."

"The world is indeed comic, but the joke is on mankind"

– Howard Phillips Lovecraft [138]

Contents

12.1 Introduction

Using ARTEMIS simulation core and the different modules previously presented, (see chapters 9, 10, 11), we built a simulation context to implement and validate the protocols presented in part III. The details of these results are presented in this chapter.

12.2 Centralized protocol

12.2.1 Flowset size

First, we wanted to focus on the impact of the network traffic on $S(\mathcal{N})$ value. Using ARTEMIS (see part IV) and its internal task generation module, we generated different flowsets of respectively 20, 30 and 40 flows. We considered a dual-criticality network, with LO and HI levels. We wanted to measure the value of $S(\mathcal{N})$ in case of a change from LO to HI criticality level. During our simulations, we based our work on the topology described in figure 12.1. We ran then simulations in a dual (LO, HI) criticality level network.

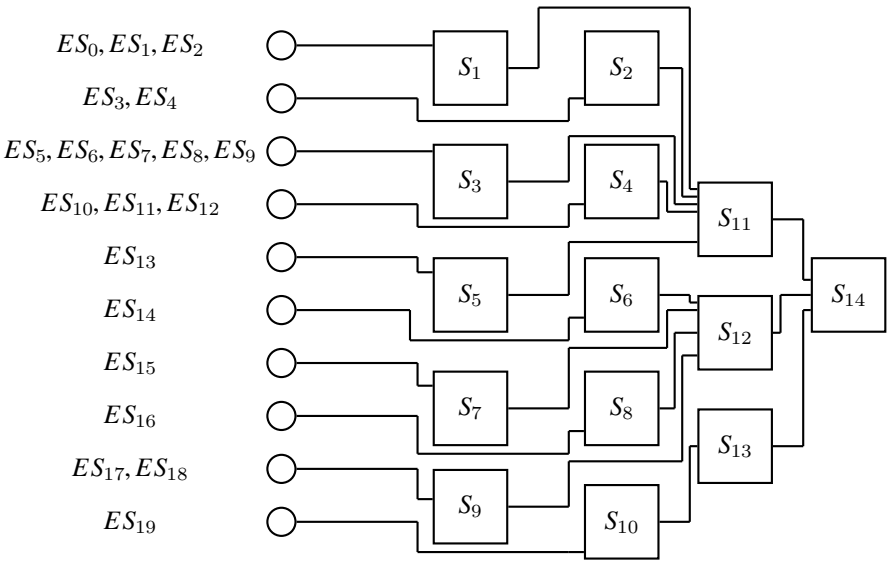


Figure 12.1: Simulation topology

We computed the end-to-end transmission delay of a HI-critical message transmitted during the transition phase.
In order to apply the Trajectory Approach, we worked on the topology decribed in figure 12.1 respecting the tree-oriented topology constraint. This topology has been automatically generated and is composed of 20 linked by a set of switches.

The depth of this topology is 4. This is obtained by computed the maximum distance $\max_i(d_n^h)$ between the central node h and the node n.
We observe that $S(\mathcal{N})$ increases when the number of flows in the network decreases, even for an equivalent load. This phenomenon is explained by the following point: the lower the number of flows, the higher the individual generated utilization for each flow. Given that the utilization is directly responsible for WCAT computation (see chapter 14.1 for more details on this point), it means that the number of flows has a direct impact of the WCAT of each flow.
The difference between the computed delays with different flowset sizes are due to the non-preemptive effect, induced by high values of WCAT. This is the results we observe in figure 12.1: the lower the number

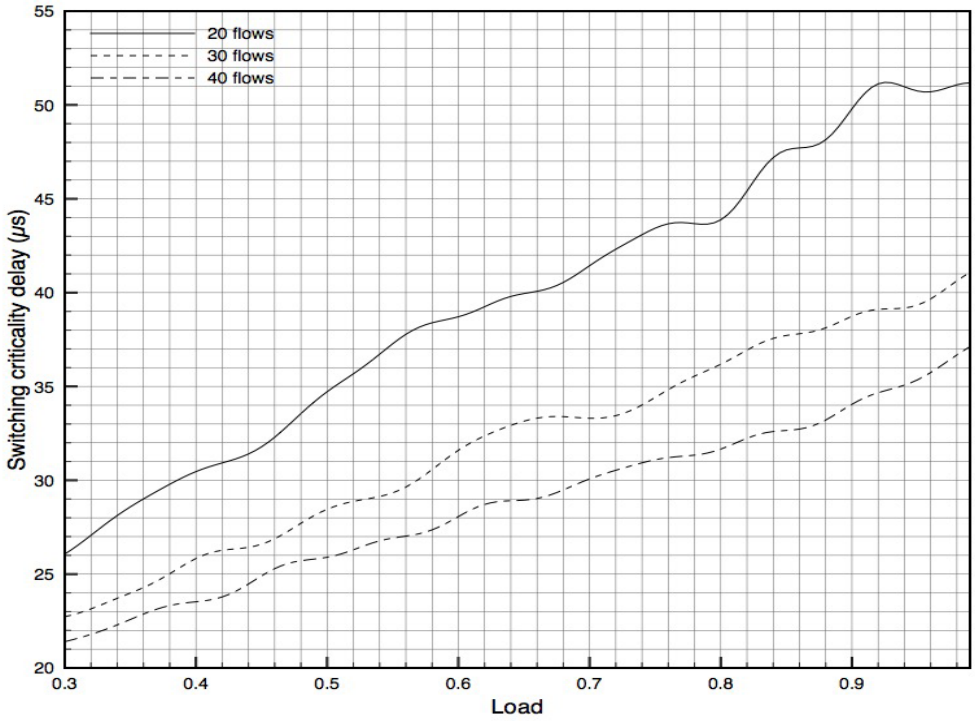


Figure 12.2: Change criticality delay with different number of flows

of flows, the higher the risk to increase the non-preemptive effect applied on $S(\mathcal{N})$. This is coherent: if we target the same load with a lower number of flows, the average and highest WCAT will tend to increase. We wanted to confirm this result, so we ran another set of simulations on the same topology in order to evaluate the impact of the highest WCAT in the network. This is the simulations detailed in the section below.

12.2.2 Highest WCAT

In terms of flowset generation, the generation of each WCAT is centered around a value which depends on 3 parameters: the number of flows in the network $|\mathcal{V}|$, the global load of the flowset l and the simulation time T. We observed the results on a $T = 500\mu$s window. According to the algorithms described in chapter 11, the generated WCAT is computed according to a uniform law.
In order to evaluate the impact of the highest WCAT, we ran 5 different simulations, respectively limiting the WCAT to 5, 7, 10, 12 and $14,47$ (Ethernet 100Mb/s limit) value. We are working in an Ethernet 100Mb/s architecture. We obtained the results described in figure 12.3.
The obtained results show what was expected: the criticality change delay $S(\mathcal{N})$ does not directly depends on the load but on the max WCAT of all flows in the network. This is coherent: the higher the highest WCAT, the strongest impact on the non-preemptive delay applyied in the computation of $S(\mathcal{N})$. The result is tough to observe on low values of the load (< 0.6). But we can clearly observe that, at high loads, $S(\mathcal{N})$ is nearly constant when the WCAT value is tight.
It means that, no matter the network traffic, the delay needed to change the criticality level is bounded.

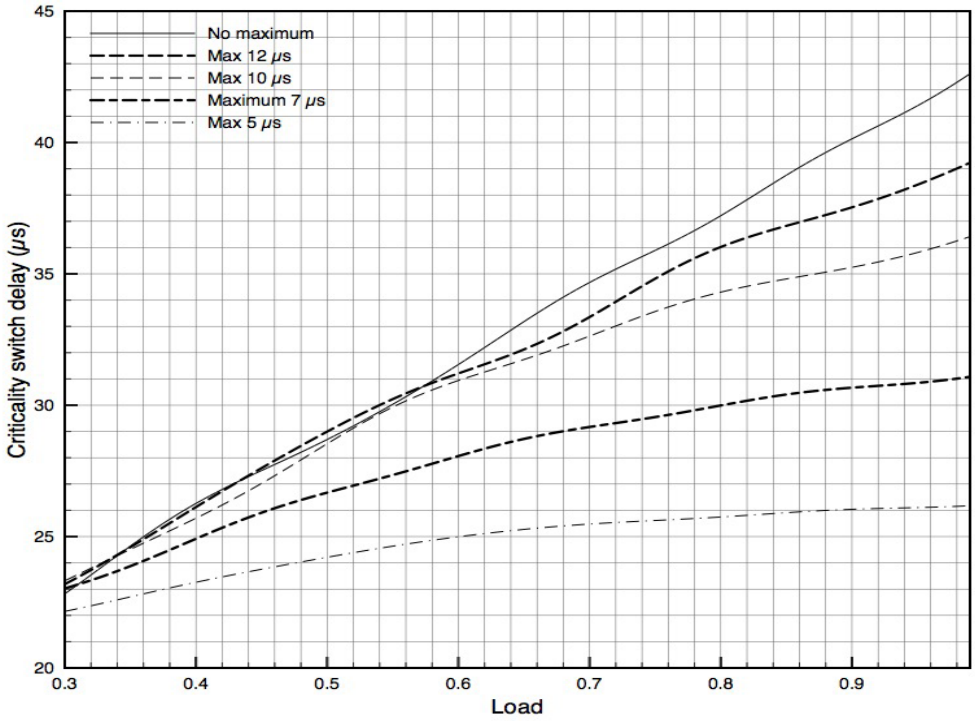


Figure 12.3: Change criticality delay depending on the max WCAT in the network

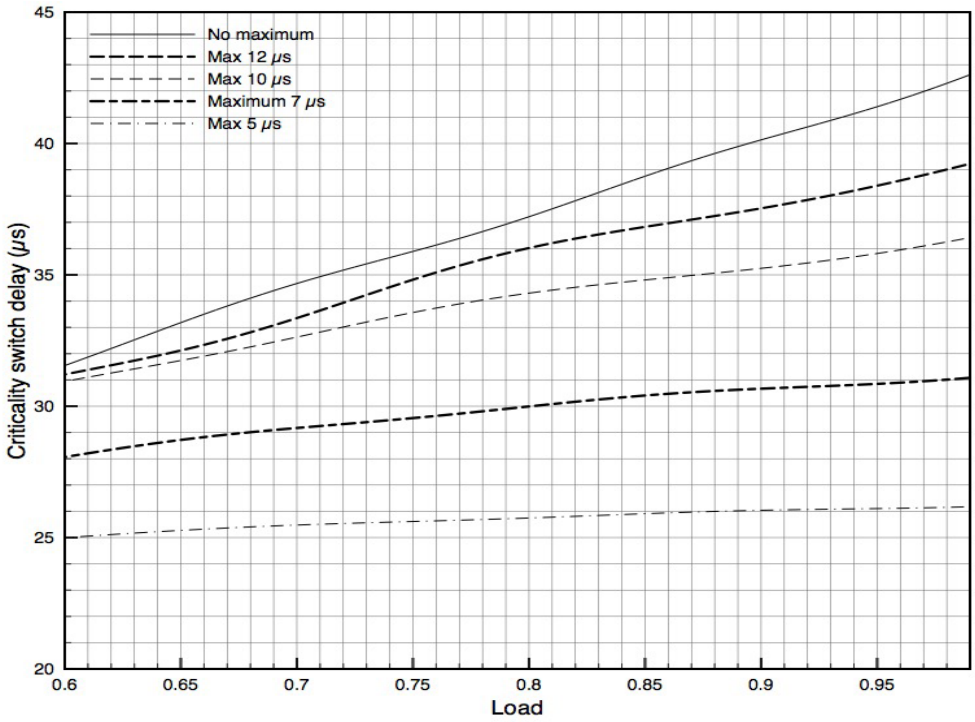


Figure 12.4: Change criticality delay depending on the max WCAT in the network (Zoom)

When the target load of the generated flowsets increases, it means that the average WCAT will increase.

But, as we limit the highest WCAT of the flowset, all the WCAT will tend to increase. As a conclusion, all the flows will tend to have a WCAT equal to the bound we fixed. It means that, in each node, the probability to have the highest WCAT of all flows in this node will tend to increase also, increasing directly the delay implied by the non-preemptive effect. This phenomenon explains that the increasing of the delay is not totally constant at high loads (see figure 12.3).

12.2.3 Blocking and non-blocking approaches

In this simulation, our purpose was to compare the end-to-end transmission delays of a γ_{new}-critical message during the criticality level change transition phase, from γ_{anc} to γ_{new}. In order to compare the γ_{anc}-critical traffic to the duration of the criticality level change delay $S(\mathcal{N})$, we based this simulation on the basic expression of the non-blocking approach, provided in section 7.3.2. We observe that the correction proposed for the delay expression of the non-blocking approach provides shorter end-to-end transmission delays.

In order to confirm the impact of both non-blocking and blocking approaches on the end-to-end transmission time of a critical message, we implemented a simulation environment with ARTEMIS. The simulation context and parameters have been defined with the different tools described in part III and IV: the topology generator (see section 11.2) and the message generator (see section 11.3).

The generated flowsets were based on several parameters: in order to generate representative flowsets and keeping the simulation time manageable, we represented random simulations of 50 and 80 flowsets.

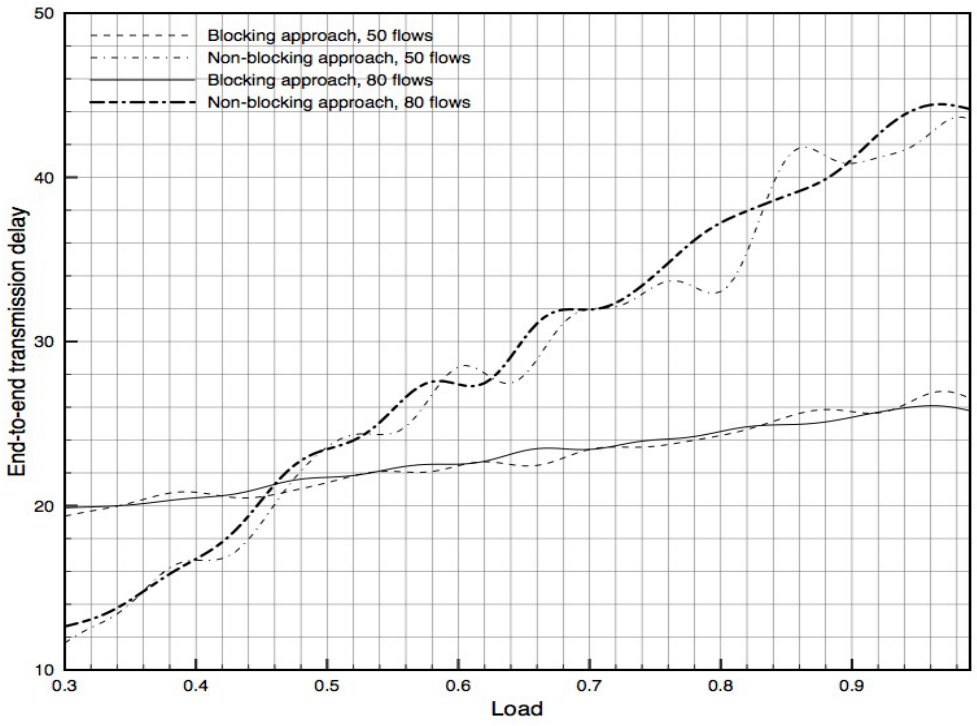


Figure 12.5: Non-blocking and blocking approaches - Impact of flowset size

We made this load increase from 0.3 (light-loaded network) to 1.0 (heavy-loaded network). Figure 12.5

shows that the average worst case end-to-end transmission delay tends to stay the same for 80 and 50 flows size flowsets. As mentioned in the previous section, we observe that the flowset size has no direct impact on the end-to-end transmission time with both approaches, as soon as the highest WCAT is bounded. In order to reduce the non-preemptive effect induced by the reduction of the flowset size, we bounded the maximum WCAT to 10μs.
This result is coherent: as the targetted load stays the same, the increasing number of messages will tend to reduce the individual impact of each one on the network traffic. Based on this result, we ran the next simulations with flowsets of 50 different flows.
Nevertheless, in real cases, the results provided by both flowset sizes are not the same. Depending on its path, the WCAT of a flow can strongly impact the network traffic, even if its value is very low according to the bandwidth. In simulation context, we suppose basically an homogeneous repartition of the traffic in the topology. In real cases, the repartition of the traffic can be erratic: high loads on specific nodes and very low on others.
In order to represent this problem and to propose a solution in the flowset generation process, we focused on the problematic of flows path computation and load repartition among the network. This particular point is detailed in chapter 11.

12.2.4 Impact of criticality configuration messages

The purpose of the simulation is to compare the transmission delay provided by both blocking and non-blocking approaches, based on different parameters. We showed with the expression 7.31 that the condition to compare blocking and non-blocking approaches was based on $S(\mathcal{N})$.
This means that the value of C_c has a direct impact on the end-to-end transmission time provided by both approaches. We wanted to verify this by simulation. Given the parameters of the task generation, we work with WCAT ranging from 0.6 to 14.47 μs in our simulation context (based on Ethernet with 100Mb/s bandwidth). In order to be compliant to these parameters, we ran the simulation with three different values of C_c: 2μs , 2.5μs and 3μs. We kept the target of 50 generated flows, with load ranging from 0.3 to 0.995.
We kept generating flowsets of size 50. The load in the network was represented by the expression $\sum_{i \in \mathcal{V}} (\frac{C_i}{T_i})$, with $\mathcal{V}$ the generated flowset. A more detailed approach about load computation in the network is detailed in section 11.3.2.
We obtained the results described in figures 12.6,12.7, 12.8 and 12.9. We observe that the end-to-end transmission delay provided with the non-blocking approach is lower than the delay provided by blocking approach at low value of loads (0.3 - 0.5). This first assumption is coherent: at low loads, the delay induced by network traffic is lower. So is the delay computed for non-blocking approach.
The condition verifying $R_i^{NB}(LO, HI) > R_i^B(LO, HI)$ is indexed on C_c value. The higher C_c, the higher load needed to reach this condition. We observe in the obtained results that this condition is verified at a load equal to 0.4 for $C_c = 2\mu$s, and at a load equal to 0.54 for $C_c = 3\mu$s.
As a conclusion, the higher C_c, the higher the load value where non-blocking approach starts providing longer end-to-end transmission delays. That is coherent with our assumptions: $S(\mathcal{N})$ value increases with C_c, and so the load represented by $\sum_{\substack{j \in \vec{\mathcal{P}}_i \\ j \in \{1,2,\ldots,n_f\}}} (N_{i,j}^{first_{i,j}} * C_j^{\gamma_{anc}})$ has to be higher to exceed this value

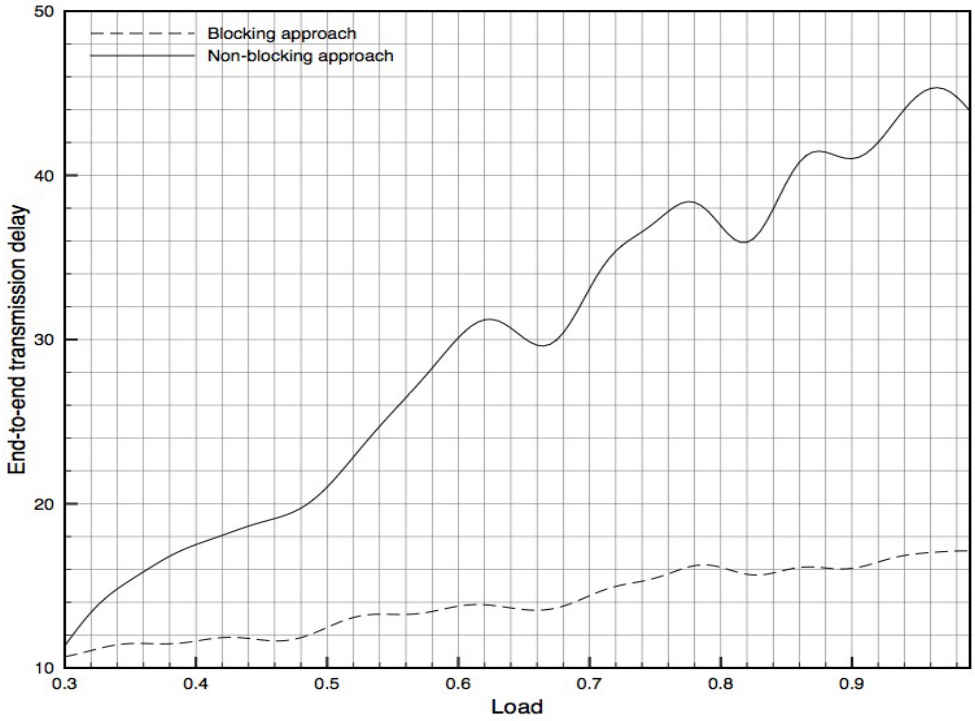


Figure 12.6: End-to-end transmission delay with $C_c = 1\mu$s

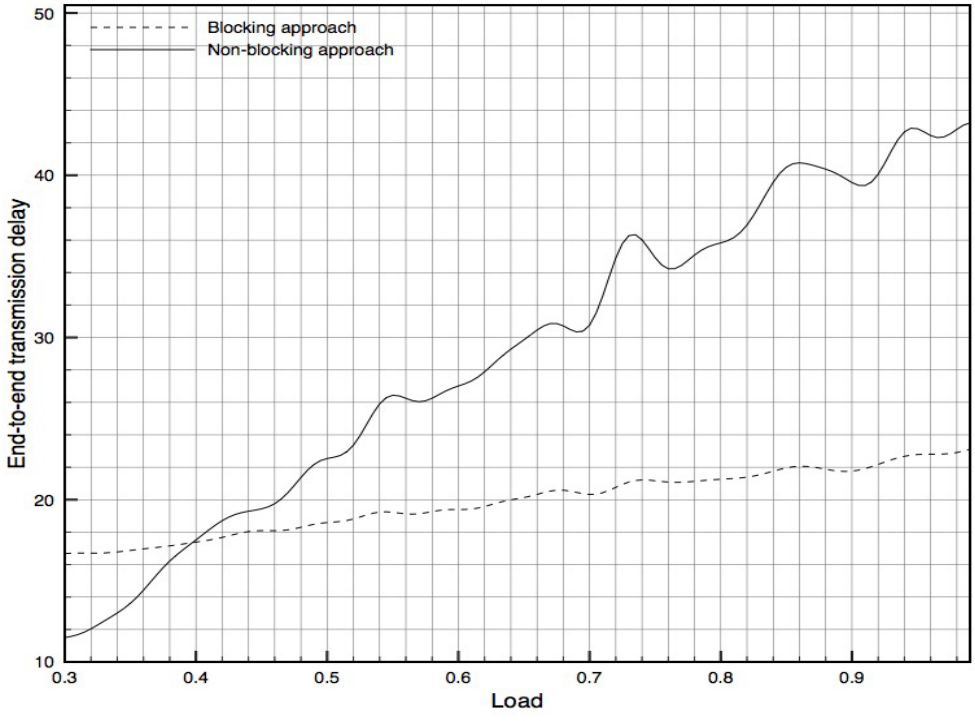


Figure 12.7: End-to-end transmission delay with $C_c = 2\mu$s

and respect the condition 7.31.

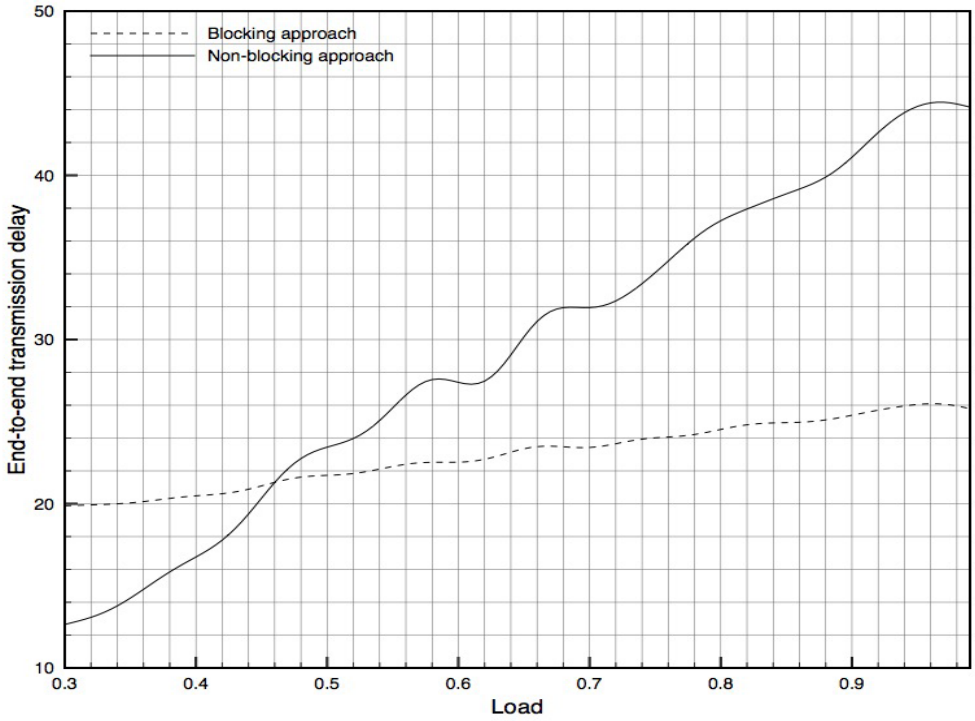


Figure 12.8: End-to-end transmission delay with $C_c = 2.5\mu$s

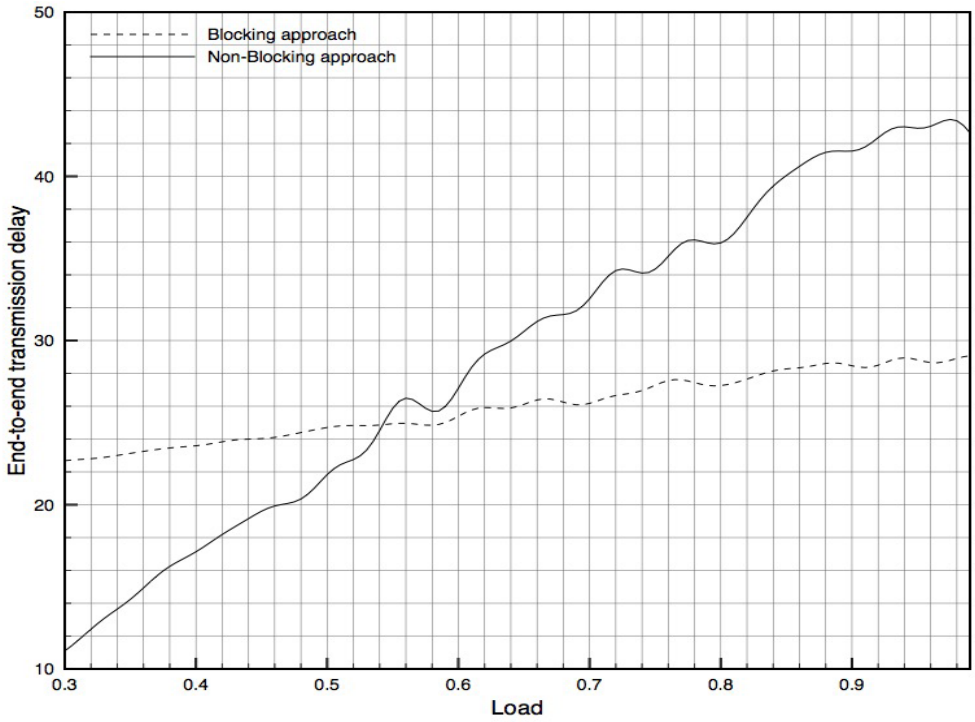


Figure 12.9: End-to-end transmission delay with $C_c = 3\mu$s

But criticality management and weak temporal isolation guarantees become particularly important to

avoid node overloads, which supposes a high global network load. In the case of a high amount of network traffic, the blocking approach provides shorter worst case end-to-end transmission delays.

12.2.5 Criticality rate

When there is a need to change the criticality level inside the network, there is not necessarily one unique critical message in the network. There can be several of them, inducing additional delay on $R_i^B(LO, HI)$, as shown in section 7.3.1. Based on generated flowsets of 50 flows, we modified the proportion of HI-critical messages compared to LO-critical messages. This allows us to evaluate the impact of the HI-critical traffic on the transmission of a HI-critical message during criticality level transition phase.

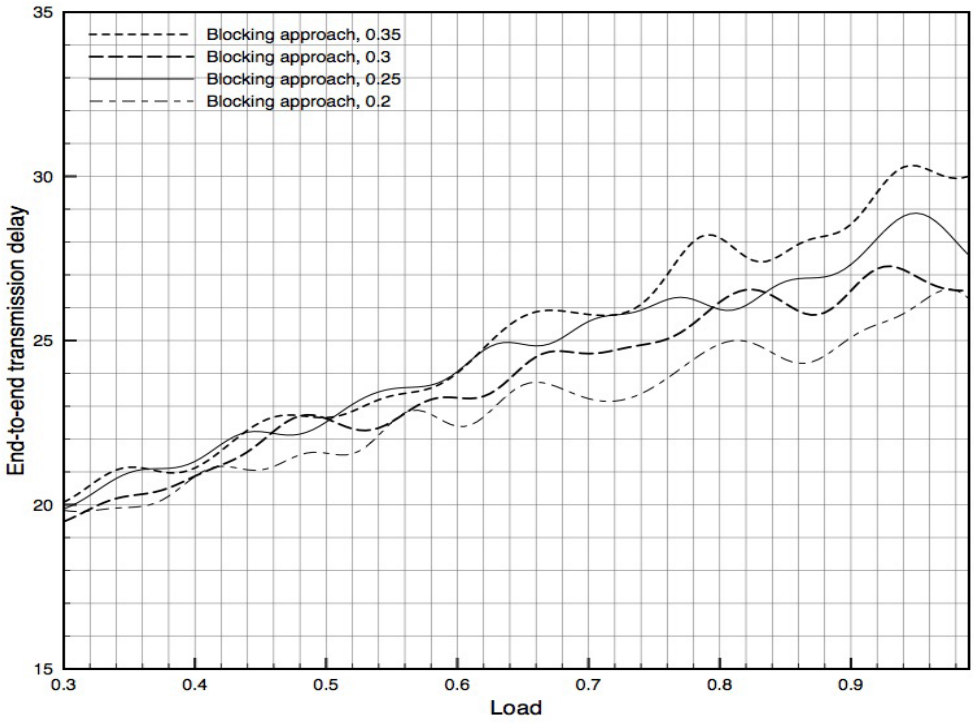


Figure 12.10: Critical rate impact on blocking approach delay

We show the simulation results on figure 12.10. The proportion of HI-critical messages progressively changes from 0.2 to 0.35, based on a network managed with the blocking approach. At low loads, there is no real impact. But for high values of the load (> 0.8), we can observe that the higher the criticality rate, the higher the end-to-end delay of a HI-critical message.
This is coherent: as we can have parallel transmissions of HI-critical messages starting from the instant the criticality level had been switched, this can represent a potential delay due to messages with a higher priority and non-preemptive effect. The γ_{anc} and γ_{new}-critical messages have both impacts on the end-to-end transmission delay of a message during the transition phase.

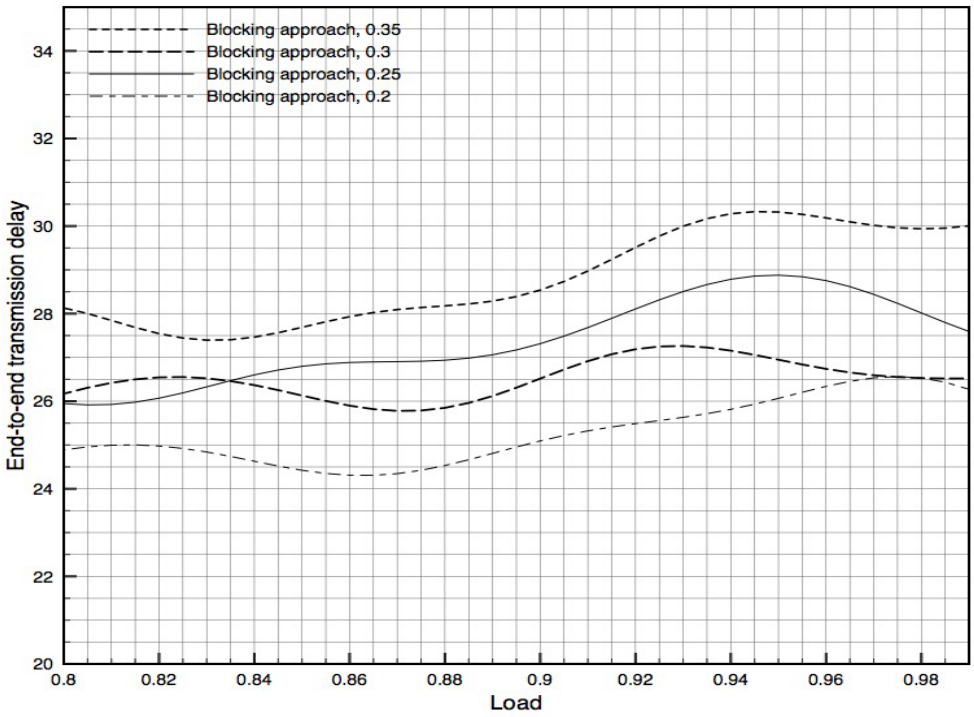


Figure 12.11: Critical rate impact on blocking approach delay (Zoom)

12.3 Decentralized protocol

For LO-critical messages transmission, the worst case situation happens when all switches detect a HI-critical message, meaning that all switches will change their local criticality level at once. In this situation, there will be no transmission of any LO-critical message. In this situation, both centralized and decentralized protocols provides the same behavior.
Our purpose was to compare centralized and decentralized approach from two different aspects: HI-critical end-to-end average transmission delay, and QoS guarantees provided by computing the number of LO-critical messages correctly transmitted.

12.3.1 Transmission delay - Static load

In terms of average delay, we can observe important differences, due to multicast and criticality change delay (in centralized protocol) which represent a non-negligible additional delay to take into account in HI-critical message transmission.We want to compare both MC management approaches for randomly generated scenarios.
In order to compare average transmission delay in both protocols, we simulated tree-oriented topologies composed of at least 20 end-systems, interconnected through a variable set of switches. The number of switches cannot be directly set as it depends on the defined network density (ratio between the number of nodes and number of switches. See details in chapter 11), specified by the user (see chapter 9 for more details about topology generation in ARTEMIS). In order to preserve consistency in the network, we set

the network density of the topology generator to 0.6. This can lead to network topologies depths varying between 3 and 9, and a number of switches in a network between 4 and 15.
In terms of number of flows, we showed in chapter 9 that the simulation time directly depends on the flowsets sizes. As RT industrial networks can have to manage at least 50 flows, the solution in this context was to select a sufficiently representative number of flows. We decided to generate flowsets of maximum size of 50 flows during our simulation protocol.
The MC integration inside a RT network and the performances of each protocol depends on the amount of HI-critical messages to manage. The end-to-end transmission delays of messages inside this topology will be impacted. In order to clearly identify the impact of HI-critical traffic, we defined the concept of HI-critical messages rate, representing the rate of HI-critical messages in the network, compared to the global number of messages. During this simulation, we set a critical rate in the network at 0.4. It means that 40 % of the messages in the network are HI-critical ($C_i^{HI} \neq -1$).
We ran 20 different topologies and flowsets per potential network size (from 15 to 120). We first simulated the computation of transmission delay of a message with a path of static size (up to 4 nodes), in order to evaluate the results given by both approaches on a fundamental hypothesis. The results are given in figure 12.13.
This simulation set was ran with flowsets of a global load of 0.85, with a cumulated utilizations of all flows equal to 0.85. The individual repartition of this load depends on the number of switches. The higher this number, the lower the average individual load on each node.
We observe in figure 12.12 that the transmission delay computed with the decentralized approach tends to be inferior to the transmission delay provided with centralized approach. In the centralized approach, the criticality level switch delay is an additional source of delay which impacts HI-critical messages end-to-end transmission time.
Thus, we can note that with a constant load, the transmission delay of both approaches is nearly constant. This tends to make us suppose that there is a balance between two phenomenons: the decrease in the transmission delay due to lower individual load per node and the additional delay required to change and manage the criticality level in the network.
As a first conclusion, we observe that the transmission delay computed with decentralized is shorter in average cases. Thus, a first observation on the figure 12.13 tends to show that the transmission delay increases with the network size. We want to confirm these assumptions.

12.3.2 Transmission delay - Evolutive load

The centralized protocol clearly depends on the network size in terms of HI-critical messages end-to-end transmission delay. The reliable multicast delay is based on the network depth (see chapter 7). This can be explained easily: the wider the network, the higher its depth. As the depth increases, the reliable multicast delay tends to also increase.
In order to confirm the results of the simulation of figure 12.13, we ran another set of simulations as a function of the network size, with a varying network load. For each value of the network size, we simulated different flowsets of loads ranging from 0.2 to 0.95.
We can observe in figure 12.13 that, when keeping the same path in all simulations, the delay given by the centralized approach tends to increase with the network.

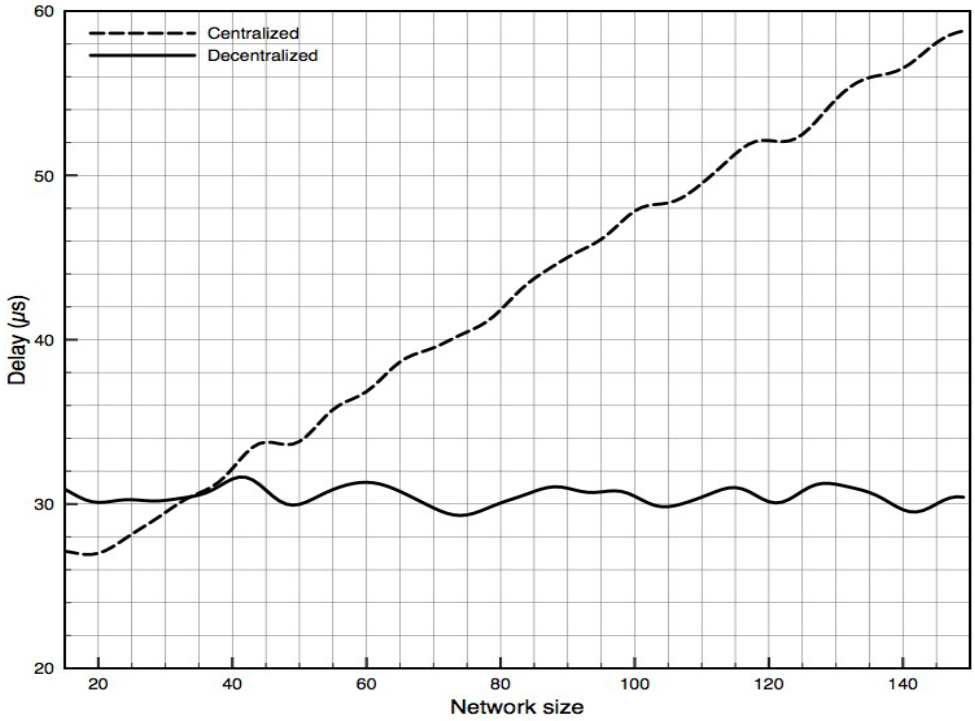


Figure 12.12: Centralized and Decentralized transmission delays function of the load

This confirms what we assumed in our theoretical work: in small topologies, the criticality change delay obtained with the centralized approach tends to be short, and so the transmission of HI-critical messages is quickly isolated from LO-critical traffic. Induced delay due to LO-critical traffic is reduced. On the contrary, when the network size tends to increase, the criticality change delay tends to increase too. This implie the transmission delay provided by the centralized approach increasing as well.
Given that the message we focus on has a static path, the delay established with the decentralized approach does not depend on the number of nodes in the network, but just on the number of nodes in the given path.

12.3.3 Impact on QoS

We focus on LO-critical traffic management with centralized and decentralized MC protocols. We compare the number on LO-critical messages correctly transmitted during both LO and HI critical phases.
In terms of QoS, we ran a set of simulations inside the same context in order to evaluate the QoS guarantees obtained by the decentralized protocol. In worst case situations (all messages in HI-mode with HI-WCAT, and a HI-critical message transiting through each node), both centralized and decentralized protocol lead to the same transmission delay. As a conclusion, in the worst case, there is no LO-critical messages transmission with both protocols (during HI phases). It means that, in worst case situations, there is no QoS provided to LO-critical messages.
In order to evaluate the average QoS provided by this new protocol (the average number of LO-critical messages correctly transmitted), we ran a simulation to compute the number of LO-critical messages

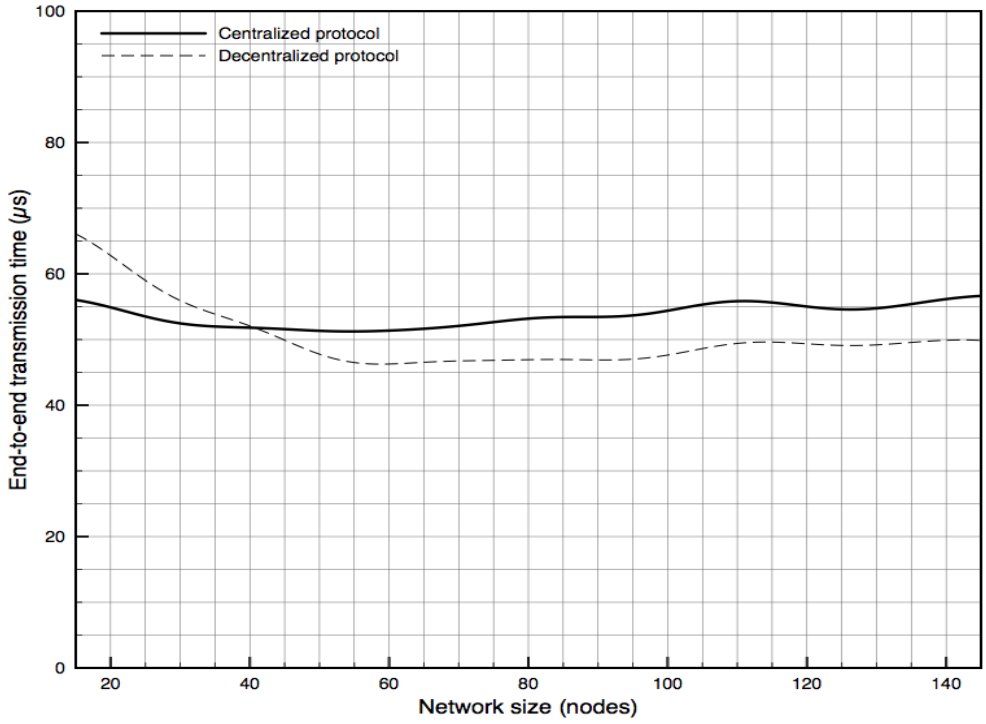


Figure 12.13: Centralized and Decentralized transmission delays

that were supposed to be dropped out with the centralized protocol, but correctly transmitted with the decentralized protocol.

In a dual-criticality network topology, the QoS obtained for the transmission of LO-critical messages directly depends on the number of HI-critical flows. This traffic is indexed on two elements. First, it depends of the global network load. It seems obvious that, the higher the global network load (LO and HI), the higher the HI-critical traffic. This traffic also depends on the repartition between LO and HI traffic inside the network.

In order to evaluate the QoS of a network configuration, we ran a simulation with different values of criticality ratio (from 0.1 to 0.4). The criticality ratio is an index of the proportion of HI critical messages compared to LO critical messages in the network (see details in 11.3.3). During all simulations, we supposed that there was, at least, one HI-critical message exceeding its LO-WCAT. It means that the provided results show the LO-critical messages transmitted during HI-critical phases.

We ran this QoS computation simulation in a network composed of 25 end-systems, with a network density of 0.6. In order to have a homogeneous repartition of the traffic among the topology, we increased the size of generated flowsets to 120 flows. This allow to identify more clearly the impact of individual messages utilizations, as the global load distribution over the network is more heterogeneous. We obtained the results showed in figure 12.14.

We observe that the higher the criticality rate, the lower the number of LO-critical messages which transmission has been assured. We also observe that, for a static load, the rate of correctly transmitted LO-critical messages tends to stabilise around 20 %, even for high values of the criticality rate. It means that, while there is no global HI-criticality traffic everywhere in the topology, we can assure at least a gain of

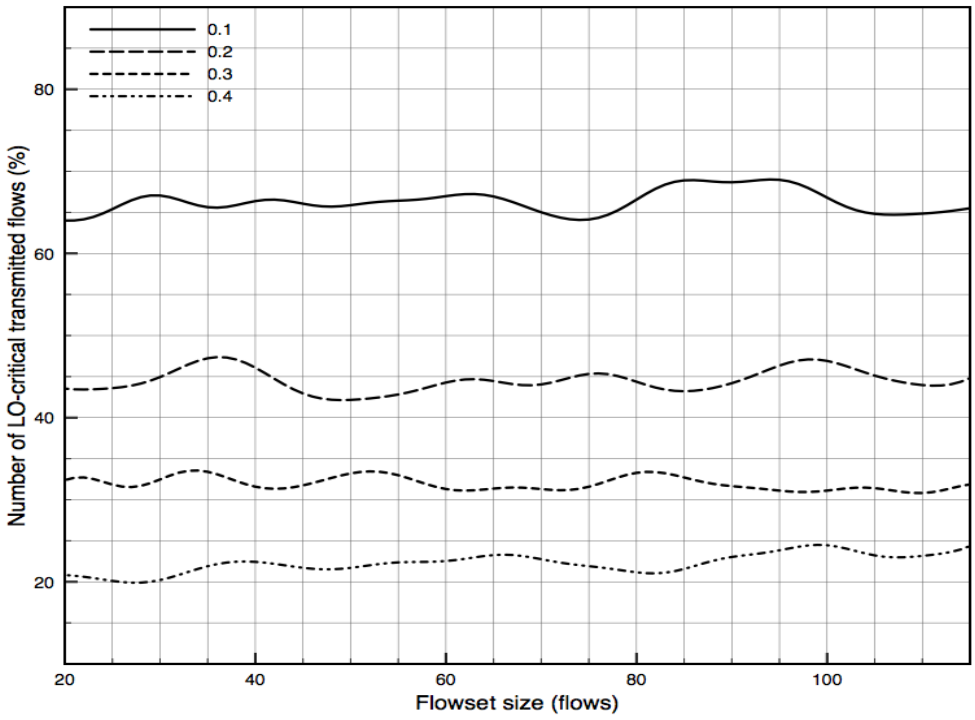


Figure 12.14: LO-critical messages transmission with decentralized protocol

QoS in terms of LO-critical messages transmission of approximately 20 %.
This simulation shows that, on wide and very wide networks (> 50 nodes), the decentralized protocols isolates specific parts and subnetworks in the topology without impacting the QoS of the whole network. Particularly in the case of a high number of flows, this allows the protocol to assure the transmission of LO-critical traffic even when HI-critical messages are transmitted without impacting the isolation constraints needed by HI-critical messages transmission.
In order to assure this gain of QoS, we ran a set of simulations to compute the number of LO-critical messages transmitted during a certain interval of time, splitted between LO and HI critical phases. There are the results of the simulation shown below (see figure 12.14).

12.3.4 QOS computation

In this context, we fixed a simple topology composed of only 5 end-systems. Working on a small topology allowed us to accurately maintain load repartition and homogeneity of the paths when generating random flows. For each simulation (50 per value of the criticality rate, with a step of 0.01), we generated a random flowset of 40 different flows, indexed on a global load of 1.0.
We ran a network simulation with ARTEMIS core during a time interval of 10s for each simulation. This delay is sufficient to be representative of, at least, on a hyperperiod. For each flow, we computed the number of messages correctly transmitted during simulation. In order to evaluate the impact of the MC management, we made the criticality rate of the network varying between 0.01 and 0.5. We finally obtained the results shown in figure 12.15

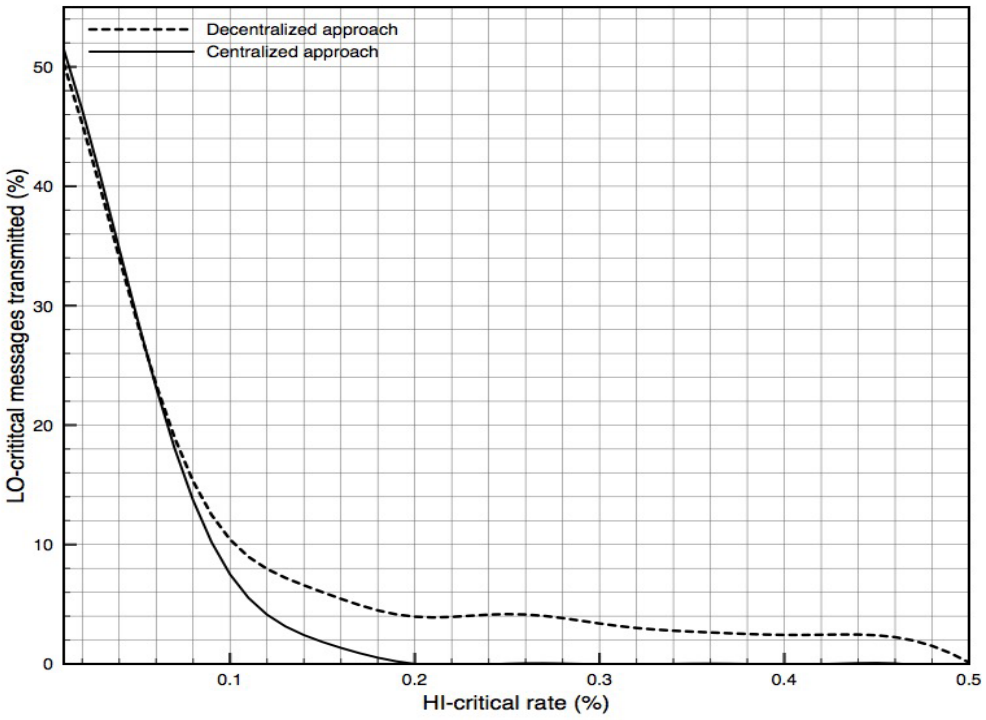


Figure 12.15: LO critical messages assured transmissions

The simulation clearly shows that the number of LO-critical messages correctly transmitted (not dropped out) is higher with the decentralized approach. Depending of the flowset and path repartition, the repartition of the flows can be different. This repartition induce criticality level changes which are likely to be more frequent, and each critical phase to be shorter.
triggering a variable number of criticality level switches during each simulation.
We observe that, when the criticality rate exceeds 20 %, the centralized approach tends to ignore all LO-critical traffic. On the contrary, decentralized approach tends to assure part of LO-critical traffic transmission up to a criticality rate of 40 %.

12.4 Conclusion

The simulation results confirms the different assumptions that were made : the decentralized protocol provides a solution to transmit a part of non critical messages, even during critical phases. Moreover, decentralized protocol provides better end-to-end transmission delays in average cases (identical in worst case studies). Eventually, for centralized protocol, the provided results shows that the criticality level change solution provided is not dependant from the network load.

Part V

Conclusion and perspectives

Chapter 13

Conclusion

"Un jour pendant lequel vous avez appris quelque chose n'est jamais complètement perdu."

"A day in which you learn something isn't a complete loss."

– David Eddings [139]

Contents

Mixed criticality management into embedded network architectures: application to switched Ethernet networks has been treated step by step. First, we proposed a theoretical solution to model MC and two protocols to manage MC in RT networks. Then, we described an implementation solution inside Ethernet networks. Eventually, we simulated these solutions through ARTEMIS. Our work is a bridge between processor and network context.

13.1 Conclusion

MC in RT networks

The first step in our work was to provide a modelling for mixed critical flows in RT networks. We had to adapt MC concepts to RT networks, from the modelling of MC in processor context given in the state of the art. We modified the fundamental representation of a flow (path, WCAT, period) by considering two potential solutions. One based on dedicated WCATs and one based on dedicated periods for each criticality level. In both cases, these solutions lead to the same integration proces. This modelling can be considered the basics of all our work on MC.
Ressources costs and weak isolation (isolation of flows from different criticality levels without physical isolation of the infrastructures) constraints were two of the major issues justifying this work. By proposing centralized and decentralized MC management protocols, we offer solutions to potential network designers to integrate MC in RT networks. We proposed different solutions in order to adapt these protocols to various contexts, making them configurable (blocking/non-blocking approach, full-centralized or half-centralized MC management). These solutions create different degrees of configurability between a full-centralized solution (requiring switches supporting clock-synchronization) and a totally distributed solution. The distributed solution, as it does not require any clock-synchronization, can be implemented in more common switches, with less configurability (cheaper, highest availability).
Simulations results show that the centralized protocol is more convenient for small network topologies, having a small number of switches. This is a protocol particularly designed for networks integrating a central node (tree-oriented topologies). In that case, we can define a node which can operate as a central MC management node. We proposed a reliable multicast solution to guarantee a total order in MC level updates all over the network nodes. This reliable multicast requires clock-synchronization.
The decentralized MC management protocol we proposed is more adapted to wide topologies and does not require clock-synchronization. Each switch has to be configured independently, but there is no additional communication for MC management required between the network nodes. This solution provides better QoS management for flows with a low criticality level. As a result, this solution is more convenient for COTS Ethernet networks, requiring high levels of QoS such as comfort management functionalities in personal vehicles, or passenger information in aircrafts.
How can we represent and manage MC inside RT networks? The proposed protocols, based on the flow representation we introduced, answer to this question. According to centralized and decentralized solutions, we can manage MC level changes and guarantee criticality level consistency inside a RT Ethernet network.

Implementation of mixed criticality in switched Ethernet

Nowadays, classical Ethernet presents a cheaper solution for RT networks compared to industrial architectures such as AFDX or TTEthernet. Additionnally, it is a standard in all network communications, even for non-industrial uses. In this work, we proposed to implement centralized and decentralized MC management protocol inside Ethernet networks. This solution relies on the classical frame modelling of Ethernet to store MC information in the frame. This solution also allows network designers to represent the criticality level of each message in the frame itself.
In order to model and transmit MC configuration information, we proposed a frame modelling for MC management messages. This frame modelling is used in centralized protocol, as decentralized protocol does not rely on any specific MC management message. This modelling is based on a specific encoding of Ethernet frame, based on the IEEE 802.1Q tag. More generally, the protocol we proposed uses IEEE 802.1Q tag to manage criticality levels information inside network frames.
We propose two different solutions for MC integration in Ethernet, relying or not on clock-synchronization. In clock-synchronized networks, this solution is based on integrating MC information inside clock-synchronization frames. We proposed an implementation of this method with PTP frames. In non clock-synchronized networks, providing solutions for multi-criticality levels integration can appear to be potentially costly (partial utilization of Ethernet data frame). But, in all cases, we provided solutions for MC management inside RT Ethernet, even for networks integrating more than 2 different criticality levels.
How can we concretely implement MC in Switched Ethernet? By using the protocol we proposed, we can integrate MC management inside a Switched Ethernet architecture.

Simulation

ARTEMIS is a cornerstone of our work in this thesis. The work done with ARTEMIS has two parts. On one side, we worked on building a RT network simulation platform. On the other side, we did a specific work around the different modules and, more particularly, on the flowset and topology generation algorithms. Both aspects of the tools are fundamental in our work, for different reasons.
Our purpose was to cross analytical and engineering approaches when developing the tool. When developing ARTEMIS, the functional and design constraints (modularity, ergonomy, ...) were anticipated in order to make the tool easier to maintain. The data API used as input and output can be improved in the future but, for now, the structure makes the simulation core independant from the different modules. As a result, each module of ARTEMIS can be used independantly.
The work done on ARTEMIS simulation consisted in building a tool to simulate RT networks scheduling scenarios. Then, we integrated MC management inside this model. In terms of simulation, MC management solutions integrated inside ARTEMIS core are functional. The tool is entirely able to manage network simulations with one, two or more criticality levels. We proved with various simulations that the model we defined was able to change criticality levels either globally or locally for each network node, corresponding to the different MC management protocols we designed.
First, the different protocols (centralized, decentralized) are represented in ARTEMIS simulation model. ARTEMIS has been fundamental in our work to experiment the theoretical model proposed and to run

different simulations on our work.
In a second part, the topology and flowset generation algorithms can be configured depending on different parameters. Each of the generation module can be used as standalone or integrated inside ARTEMIS whole suite. In the case of flowset generation algorithms, being based on UUnifast assures a potential standardization in the flow generation process. Both algorithms allowed us to proceed simulations on wide number of potential situations. It allowed us to run the different simulations used for MC integration protocols, to verify our solutions, etc... on representative sets of situations. Thus, these tools proposes improves of actual generation models currently used in processor context.
ARTEMIS development is not supposed to end after this work. During our work, we developed a software solution to answer to MC problems, but the open and modular structure of the tool encourage us and external development teams to improve the tool, either by adding new functionalities or by modifying its usability and integrability.
How can we build software and hardware solutions to verify our work? ARTEMIS is the simulation solution provided to run MC scheduling scenarios inside RT networks.

General conclusion

To conclude, MC can be integrated in RT and embedded Ethernet networks through dedicated protocols, assuring the timeliness and safety of messages transmissions inside a network. Corresponding to a correct message tagging and to centralized or decentralized MC management, each node in a network can trigger a criticality level change along time. The conditions to change the criticality level of a node will depend on the criticality of each message and on the messages likely to be transmitted through the node.
So, What is MC applied to Switched Ethernet networks? MC management is a solution to assure the reliability, safety and security of an embedded network during critical phases. These critical phases are defined by specific periods when the network input and output starts dealing with critical messages. This can be due the message length evolution, or to the IEEE 802.1Q tag indicating the criticality level of the message.
Critical phases corresponds to tough or demanding situations, when the transmission of all messages cannot be guaranteed, implying to privilege critical messages among all messages. We guarantee, through the solutions we proposed, a weak temporal isolation between critical and non critical informations. That is what MC management is: mixing messages of different criticality levels inside the same infrastructure while guaranteeing temporal isolation and timeliness.

13.2 What's next? Perspectives.

Industrial implementation

Industrial platforms

The MC management protocols we proposed were validated by simulations with ARTEMIS. But from now on, MC inside networks has not been implemented yet on physical platforms. Providing an evolu-

tion to test MC integration inside Ethernet-based embedded networks would represent a strong physical proof of concept of what we presented in this work. Through various real-time operational layers (such as Xenomai), building an experimental platform to test and validate MC integration solutions represents an interesting solution to work on.
The centralized and decentralized protocols provides implementable solutions for various network platforms, at various costs. By proposing different protocols applicable to clock-synchronized and non clock-synchronized networks, we proposed solutions which can fit either to safety-critical network Ethernet infrastructures or to COTS Ethernet infrastructures. But all the solutions we designed still requires a certain level of configurability. It cannot necessary be found in all switches, particularly the cheapest one. An interesting potential perspective would be to propose appliability of MC on very basic switches. This will induce to open MC integration to non deterministic networks.

Hypervising

Assuring different levels of QoS inside a network is an important requirement. Different application domains, particularly those related to Internet applications, rely on the objective of bringing a high level of usability and QoS to the user. We can mention, for example, domains such as online gaming, web applications or home automation. Software-Defined Networks (SDN) [140] are a solution to configure these networks by creating an intermediate abstraction layer, virtualizing the network management and offering QoS management solutions through APIs. We can mention different tools to integrate this abstraction of physical switch management, such as OpenVSwitch [141].
One perspective of evolution for RT networks would be to integrate MC management inside SDN solutions. This will allow the user to manage criticality levels of switches through an API and dedicated programmation languages. This can be presented as a solution to avoid an individual and manual configuration of criticality management parameters for each network switch. The criticality implementation inside a network would become global and easier to manage.

MC improvements

Heterogeneous networks

The integration of MC has been proposed for Ethernet-based infrastructures, and more precisely for IEEE 802.1 Ethernet. But the emergence of various new network infrastructures (wireless networks, for example) or other industrial infrastructures could require MC integration inside their systems. A potential evolution of our work would be to propose MC integration solutions for alternative network architectures.
Based on this integration for different network infrastructures, a potential evolution of this work would be to propose MC integration for networks mixing different physical implementations. For example, proposing MC integration for multi-hop networks would represent an interesting evolution of the current work.

Configurability and COTS Ethernet

The MC integration protocols we proposed represents solutions for MC integration inside various Ethernet implementations. These implementations are oriented to different type of devices, from COTS to high-constrained devices based on industrial protocols such as PTP. But, in all cases, the protocols we proposed required a high amount in the configurability of the network devices, which could represent a problem of costs and availability.
A potential evolution to this would be to propose a discrete integration of MC inside Ethernet networks which will not require specific configuration of the network. This solution can be implemented, for example, by providing MC-compliant drivers and network protocols inside basic commercial devices.

Multicore switches

One major hypothesis in our work has been to consider that switches in Ethernet networks are all based on a monocore architecture, which makes them able to process one message at a time. This approach is reliable, but incomplete. One interesting perspective in the potential future works issued from this work would be to operator on switches integrating multicore architectures. This would represent a crossover between RT networks and RT multi-processor platforms.

Simulation tools

Generation tools

In this work, we proposed different solutions to generate network messages for RT network simulation. These solutions are all based on different adaptations of the model provided by UUnifast: uniform random generation, integration of different criticality levels, etc... These models are integrated in tools like ARTEMIS and can be reused in the future. But, on potential perspective of development around this issue would be to focus on how to provide network traffic generators whose results are based on physical experiments.
When defining constants such as criticality rate or network density, one source of evolution would be to provide solutions to evaluate these configuration parameters in order to provide messages generation tools which results are more representative of physical implementations. In order to do this, we have to focus on the different solutions used to define the different criticality levels of each message in a network, for example.

ARTEMIS simulation

ARTEMIS can be improved through various approaches, depending on the aspect we want to develop. We can mention some potential sources of evolution:

- Simulation model: The modelling of a network is currently generic. The node-link-flow modelling can apply to any network scheduling scenario, but it has no specificities to represent dedicated physical situations. One major source of evolution would be to propose a virtual layer to model

dedicated hardwares in the tool. This will integrate the representation of specific devices, allowing the user to fit his simulations to more specific industrial contexts.

- Network tool: At the moment, ARTEMIS is a simulation software with no interaction with real networks. An interesting source of evolution would be to propose to make it as a network virtualizer. The point would be to connect the network topologies simulated by ARTEMIS to real network infrastructures. ARTEMIS will then act as a network traffic manager and would be able to integrate functions like monitoring.

- Modules: several external modules of ARTEMIS (GUI, Grapher, Analyzer) are dedicated to results exploitability and ergonomy. These modules can be improved by improving their configurability and the precision of their results. The timing analyzer or the grapher could be modified in order to provide more detailed results. This will improve the accuracy and ergonomy of the tool by providing more detailed results, and to provide them quicker.

ARTEMIS is entirely open-source. It means that all these improvements can be brought either by its current developers, or by new development teams. ARTEMIS is currently available in public platforms to download and modify. For our part, we will continue to improve it in the future, but we also strongly invite external actors to participate to the development with us.

13.3 Personal perspectives

All along this thesis, I focused on MC integration inside RT and embedded networks, and more particularly in Switched Ethernet networks. All along this work, I did encounter different people, focused on different related and unrelated topics, and my participation to the topic through this thesis makes me ask about this question: so, what's next?
There is a lot of remaining open doors. RT and, more widely, computer science are boundaries which cannot explore in their whole globality in a single life. Tolkien said "All we have to decide is what to do with the time that is given us" [142]. These emerging topics are part of the different solutions I would like to explore in the following years.

Around RT

Diving into RT domain and into RT simulation tools made me focus on various software solutions dedicated to RT. There are technologies such as Ada and RT Java which appears to me as an interesting source of potential evolution. These technologies mix RT problematics and object-oriented development. It represents an interesting balance between engineering and research, and as a result I would like to learn new skills related to these topics. Thus, concerning ARTEMIS, these technologies represent sources of new improvements to integrate in the tool.
During the work around ARTEMIS development, I used to build a global data modelling solution to represent ARTEMIS input and output informations. This data modelling layer have been fully detailed all along this thesis. Coupled to several external works done with labs like Institut de Recherche en Informatique de Toulouse (IRIT), I had the privilege to work on data modelling for RT simulation tools.

That opened a potential project of providing standardized data modelling format on which I would like to work in the future. This work, initiated with actors like Claire PAGETTI, Sophie QUINTON and Loïc FEJOZ is representing for me a strong source of potential evolution.
Eventually, I would like to work on QoS management for mixed critical RT networks. The work we did with the decentralized protocol particularly directed me to ask myself how to manage non critical messages and assure QoS guarantees inside RT networks.

A few words about teaching

"To teach is to learn twice" said Joseph Joubert. During this thesis, I began to work as a fellowship teacher in computer science. That has been a real revelation to me. From my point of view, teach and research are the two faces of the same coin. During these 3 years, I used to teach various domains around computer science, development and integration. In the future, I would like to continue this by opening my research work to students.
Through publication, conferences or dedicated lessons, I would like to introduce RT scheduling and embedded networks to my future students. An important project I would like to build is to publish various teaching books about computer science and development skills. It would allow me to train pedagogical approaches while keeping the habit of writing through the time. That would be an opportunity to continue working on both research and teaching sides.
Thank you for reading.

Chapter 14

Annexes

"Ne dites pas trop de mal de vous-même: on vous croirait."

"Do not say too much harm of yourself: one would believe you."

– Amélie Nothomb [143]

Contents

14.1 Mixed criticality over Ethernet protocol

14.1.1 Introduction

Based on our previous work [124], we want to propose concrete solutions of MC implementation in Ethernet. Depending on the MC management protocol and on the number of criticality levels we want to manage, the solution to use is different. First, we introduce a solution to dual MC representation in PTP clock-synchronized networks. Then, we apply this solution to non clock-synchronized networks. Eventually, we extend this solution to multi-criticality levels.

Mixed criticality over PTP Ethernet

IEEE 1588 PTP is a clock synchronization protocol for switched-Ethernet networks (see section 3.3). The IEEE 1588 clock synchronization protocol is used, for example, by industrial manufacturers like CISCO or MOXA for integration inside various domains: defense, aerospace, etc... From the point of view of software, the PTP daemon is an implementation of IEEE-1588 protocol, mainly designed for Unix-based and embedded architectures.
PTP is based on dedicated frames used for clock synchronization. These frames are based on User Datagram Protocol (UDP), with an integration of a dedicated PTP header in the frame, used for time-synchronization. These frames are usually sent with the highest priority in the network, by attaching them to a dedicated VLAN used for configuration purposes. PTP daemon proposes two solutions: PTP-ETE and PTP-P2P modes. In PTP-ETE, only the end-systems to manage clock-synchronization whereas, in PTP-P2P, all nodes in the network manage PTP frames.
In both end-to-end and peer-to-peer modes, PTP frames are modelled following the same structure [49] (detailed in section 3.3). Briefly, this structure is organized as follows: a header of 34 bytes describing the specificities of PTP protocol, a body of varying size containg timestamp, and an optional suffix.
PTP frames are periodically sent in the network with a high priority. Criticality level information needs also to be sent with a high priority in the network, specifically in order to respect the constraints of the centralized protocol. Thus, the synchronization of criticality information in all the network nodes implies transmitting criticality information without being delayed by the network traffic. The solution we propose is, instead of creating messages dedicated to criticality information transmission, to modify the structure of PTP frames in order to integrate criticality level management directly inside them. This will define a common time reference and criticality synchronization with an already defined standard. This also assures that criticality information is transmitted with the highest priority.
The centralized protocol for MC management is based on a clock-synchronized architecture. As mentioned in chapter 7, MC is considered as a configuration information in the network. This information is shared among all nodes through the reliable multicast protocol we detailed (see chapter 7). We proposed to send MC level in a dedicated VLAN (based on Ethernet 802.1Q) attached to the highest priority (same as PTP) in order for the SCC and switch-level order messages to not be delayed by other messages in the network. The centralized protocol proposes two types of messages to manage MC:

- The SCC message, which is a call from a central node to change to a criticality level, which value is indicated in the message frame.

- The switch order, which forces all nodes to change their internal criticality level.

The two different MC management messages are transiting through the same nodes. Even associated with the VLAN with the highest priority, SCC messages can still be delayed by PTP messages. On the contrary, multicast messages are transmitted in the opposite way, from the central node to other nodes, and are supposed to not suffer from network traffic.
A solution to MC implementation was proposed in [124]. This solution is based on mixing PTP synchronization messages and MC configuration messages inside the same frame. In other words, it consists in modifying PTP frames in order to integrate criticality information inside it.
We propose here to dedicate a byte in PTP frame to specify the criticality information. With this solution, we can broadcast the criticality level frequently, but this is not entirely synchronous: as we rely on PTP synchronization frequency to share criticality information, this can imply an additional delay in MC information transmission (through reliable multicast, in the hypothesis of a centralized approach). This additional delay impacts the global delay of criticality switch, adding to it a delay equal to $\frac{1}{F_{PTP}}$, with F_{PTP} the PTP synchronization frequency.

Mixed criticality integration in PTP frames

The first point to notice is that we do not have to use all PTP messages for MC integration: the criticality information transmission does not necessarily imply to be udpated at the same frequency as PTP emissions. It means that we need to be able to tag each PTP message to determine whether or not it contains MC information. As a result, MC integration inside PTP implies defining 1 dedicated byte: 4 bits to tag the PTP frame as containing MC information, and 4 bits to store the MC information itself.

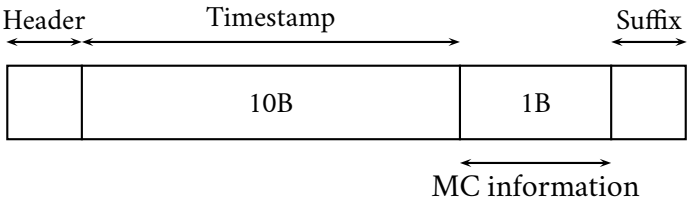


Figure 14.1: Criticality integration in PTP frame

The PTP body is composed of 10 bytes, representing the timestamp to send, used for clock-synchronization. We add a byte to this PTP body in order to represent criticality information. The figure 14.1 shows the structure of the new PTP frame.

With this solution, we can integrate criticality level inside PTP frames, but we cannot differentiate PTP frames integrating MC information from those who are not. A solution to this would be to measure the size of the PTP body: if its 10 bytes, it is a simple PTP message. If the size of PTP body is 11 bytes, it is a PTP message integrating criticality information. We need to adapt the proposed format in order to integrate these new constraints.

Mixed criticality tags

In terms of structure, MC integration in PTP frames is operated on one byte (8 bits). This byte has to contain two different informations: the type of the message, and the value of the criticality level the message belongs to.

4 bits to encode the criticality level is sufficient. The two MC description bytes are used to tag a PTP frame and to specify criticality information in the network. As a result, we obtain the PTP frame structure detailed in figure 14.2.

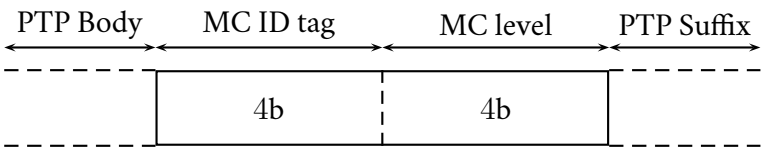


Figure 14.2: PTP body with MC integration

We propose to split the byte integrated in PTP frame in two equal parts of 4 bits. We dedicate the 4 first bits to message type designation, and the 4 last bits to criticality level definition. This allows the protocols to manage, at most, $2^4 = 16$ different criticality levels.

Each type of MC configuration message should be possible to identify (SCC or switch order). According to the centralized protocol (see chapter 7), we can enumerate two different MC management messages: A SCC or a criticality level switch order. Both messages are linked with a specific criticality level: The SCC is attached to the criticality level a specific node calls to change, and the switch criticality level order (multicasted) is linked to the criticality level the nodes have to change to.
In order to integrate MC inside Ethernet, a MC configuration message has to be tagged according to its type. We propose to attach a specific binary value to each type of message, and to encode this value into the MC-dedicated byte of PTP frames. We have 4 bits to specify this value. We propose to follow the following process:

Binary	Value	type
0000	$0x00$	Error management
0001	$Ox1$	SCC
0010	$Ox2$	Switch criticality order
–	Others	Future uses

The specification is done according to the values described in this table. Basically, we defined only two specific tags ($0x1$ and $0x2$) for the different message types we want to use. All potential other values ($0x3$ to $0xF$) could be used for potential improvements of the centralized protocol (for example, to determine if nodes should behave according to blocking or non-blocking approach while receiving the message).

This solution does not necessarily imply each PTP message to be formatted with criticality information. It means that we need to integrate the possibility to identify PTP messages containing criticality information and those which are not.

Criticality information management

In terms of scheduling, modifying PTP frames structure implies redefining the algorithm to schedule and detect MC-tagged frames. We propose an algorithm to take into account criticality information and to trigger associated actions. The potential actions associated to the reception of a MC message are:

- Update the criticality table: In the central node of the network, in case of the reception of a SCC from a lower criticality level than the current one, we node that the transmitting node sent a message to downgrade the criticality level (for centralized protocol).

- Change the criticality level: this will change the value of the integer representing the criticality level of the current node.
- Multicast a criticality level switch order (for centralized protocol).

We suppose an Ethernet network architecture composed of a set of interconnected switches. Once there is an incoming PTP message in a switch, it identifies the frame as containing criticality synchronization information or not. Next, depending on whether or not the switch is defined as the central node in the network, the node will perform different actions depending on the type of the received message. At each reception of a MC management message in a given switch, we can define the behavior of a node with the algorithm detailed below (see algorithm 11).

```
Data: Criticality message m, current
      criticality level Γ
Result: criticality level managed
b = read_4_bits(m);
γ = read_4_bits(m);
if node is central node then
    if b == 0x1 then
        if γ > Γ then
            Multicast (γ);
        end
        if γ < Γ then
            e = get_sender_of_message();
            change_sender_state(e, γ);
        end
        if all senders are in γ state then
            Multicast (γ);
        end
    end
end
if b == 0x2 then
    Wait for criticality change date
     t_m + M(N)
    Change criticality level in the node to γ;
end
```

Algorithm 11: MC message reception algorithm

When receiving a message, we read the first byte containing both criticality information and message criticality type. If the message is an SCC, its impact will be different depending if the current node is the central node or not.
In the central node, the SCC message triggers a reliable multicast emission (when $\gamma > \Gamma$) or an update of the criticality table in the other case. In an other node, SCC messages are just forwarded to the central node.
Depending on the current state of the criticality table, an update of its informations can induce a multicast message, to a lower criticality level.
When receiving a multicast message, a node received the order to change its internal criticality level. The node will wait until the date to change this level ($t_m + M(\mathcal{N})$, described in chapter 7). At the date $t_m + M(\mathcal{N})$, the node will change its criticality level.

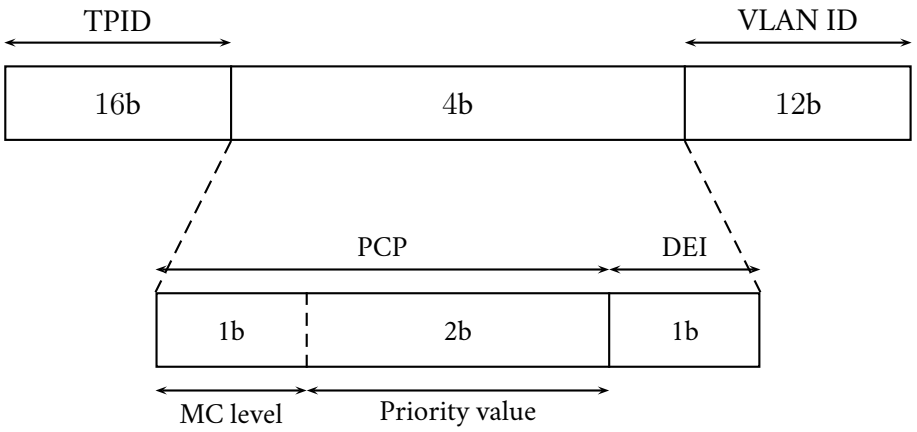
14.1.2 Criticality level integration in Ethernet

The proposed solution consisting in integrating MC management inside PTP frames can only work with clock-synchronized networks. This assures a solution to share criticality management messages required by the centralized protocol. But, this solution does not specify how we determine the criticality level of each message inside a network. Particularly in our work, we focus on MC integration in COTS Ethernet architectures. In order to do so, we provide a solution to indicate MC information of each message inside an Ethernet frame.

We propose to tag each message with its highest level of criticality, in order to determine to which level it is able to increase. This will also specify that a message can belong to the lowest criticality levels, thanks to the hypothesis about the hierarchical structure of criticality levels made by Vestal in [114] and detailed in part II. As a conclusion, the issue here remains in being able to tag each Ethernet 802.1Q message with its criticality level.

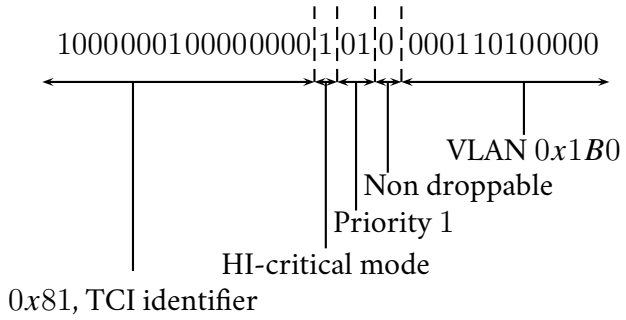
Dual-criticality level networks

First, we want to focus on a dual criticality network, characterized by LO and HI levels. In this case, a message can have two different levels: non critical (LO) or critical (HI). This can be represented by a boolean value, indicating whether the message is critical or not. The criticality level of a message specifies if a message can be transmitted or not during specific criticality phases. A HI message can be transmitted during LO and HI phases, whereas LO messages can only be transmitted during LO phases.
The criticality level for HI messages indicates that a message can have a LO and HI WCAT, and a LO and HI period (or minimum inter-arrival time). A constraint of a HI message is to have a longer WCAT or a shorter period than its equivalent LO message. Being HI-critical specifies that the message is part of the HI message set in the network, and so that its LO-WCAT can be exceeded.
In order to integrate MC management, the solution we propose is to add a field in Ethernet frames, through the IEEE 802.1Q tag, in order to tag and represent a message criticality state (HI or LO) directly in its Ethernet header. The solution is to modify the network frame of a message in order to dedicate a specific set of bits containing the criticality value. Contrary to PTP messages which are based on the IP structure, we cannot make the assumption of a specifical frame structure in our network. We showed that each protocol and layer (IP, UDP, ...) adds its own header and structure to the Ethernet frame. As a conclusion, we have to integrate the criticality information of a message at the lowest possible level of the OSI model. In order to be compliant with our constraints, we propose to integrate criticality information directly in the Ethernet frame layer.
Ethernet is based on network layer of the OSI model, on top of the physical layer. An Ethernet frame conforms to a specific format, which was detailed in chapter 3.2. We propose to add the MC level of a message inside its 802.1Q tag (see figure 14.3).
The Ethernet header is based on 14 bytes. All these bytes are dedicated to a specific role and cannot be modified. Our solution for MC integration inside COTS Ethernet is to modify the PCP field in order to use the most-significant bit to indicate whether the message is HI-critical (1) or LO-critical (0).
Using this solution, the priority assignment of Ethernet 802.1Q is only defined on 2 bits instead of 3. The figure 14.3 shows the proposed modification of Ethernet 802.1Q tag.
This implementation implies increasing the size of Ethernet frame by 4 bytes (IEEE 802.1Q tag size) to integrate MC information (see figure 14.3). It is sufficient for a basic criticality integration, as a single bit is sufficient to determine if a message is critical or not.
In order to illustrate the 802.1Q tag structure, we detail the structure of a 802.1Q Ethernet frame through an example. We suppose that an incoming message m arrives in a node n of the network. In order to focus on the criticality attribution of this message, we suppose that the frame header is composed of the following data frame: 10000001000000001010000110110000. We detail below the organization of the frame (see figure 14.4).



The purpose is to change the first of the three bits dedicated for priority value definition by MC level specification. If it is 0, the message is in LO mode. If it is 1, the message in in HI mode.

Figure 14.3: Ethernet 802.1Q tag with dual-criticality management



We can observe the different parts of the 802.1Q tag: The TCI identifier (used to indicate that the following bytes correspond to 802.1Q protocol), the criticality level indicator, the priority value bit (01), the drop elligible indicator (1) and finally the VLAN identifier the message belongs to (00110110000).

Figure 14.4: Ethernet 802.1Q header with MC

The analysis of the first 4 bits of the PCP field determines if the message is a HI-critical message. This analysis is likely to trigger a SCC emission in the node where the message arrives, during LO-critical level phases. When a node will receive this message, it will have to compute the criticality level of the message and its size, in order to compute its current WCAT and detect to which criticality level the node may call to change to.

The proposed implementation integrates dual-criticality level management inside RT Ethernet networks, even in non clock-synchronized networks. With this implementation, we can manage criticality for centralized and decentralized MC management protocols. The criticality level of each message is directly indicated in its header. But the representation of the criticality level is limited to 2 values (as it is encoded on 1 bit). In the following work, we propose a solution to extend this implementation to multi-criticality levels networks.

Multi-criticality levels

In RT and embedded networks, it can become necessary to define more than two criticality levels inside a topology. As a result, we need to propose a solution for MC integration inside RT Ethernet networks allowing us to manage more than just LO and HI levels.

This represents a conflict with the previously presented solution: a single bit is not enough to represent 6 different levels of criticality (or more). One naive solution could consist in using the 3 bits of the PCP field to represent criticality information, but that would imply to delete priority information inside Ethernet

802.1Q header, which is not acceptable.
802.1Q tag and Ethernet formats are standardized and correspond to specific standards. In order to integrate MC value management, we need 4 bit to be able to model an integer value included between 0 and 15. This value will represent the criticality level of a message.
To implement this, we use the Vestal's hypothesis [114] introduced in chapter 6. We consider a hierarchical order among all criticality levels. It allows us to associate an integer value to each criticality level, and to consider each criticality level as more or less critical than another one. With this hypothesis, we can affirm that a message of a γ_i level of criticality can be analyzed and sent in all $\{\gamma_1, \gamma_2, ...\gamma_{i-1}, \gamma_i\}$ levels. The solution to attach an integer value (representing the criticality level) to a message allows the MC management protocol to define the highest criticality message it can transmit.
The remaining problem, detailed below, is to find a specific spot in Ethernet frame to store these 4 bits.
The solution presented before cannot be extended by using Ethernet format and a single 802.1Q tag. We cannot either integrate MC information in Ethernet frame payload: it would require to modify all the different OSI layers (starting from network layer) to take into account this information. We propose to focus on a new approach.
802.1Q-in-Q [144] is a solution introduced in Ethernet to add successive tags in Ethernet headers. By using this solution, we can provide to add two 802.1q tags inside Ethernet header, and to dedicate the first to MC management. The constraint implied by IEEE 802.1Q-in-Q is to add another Ethertype in Ethernet header in order to distinguish the different tags in Ethernet header (see figure 14.5).

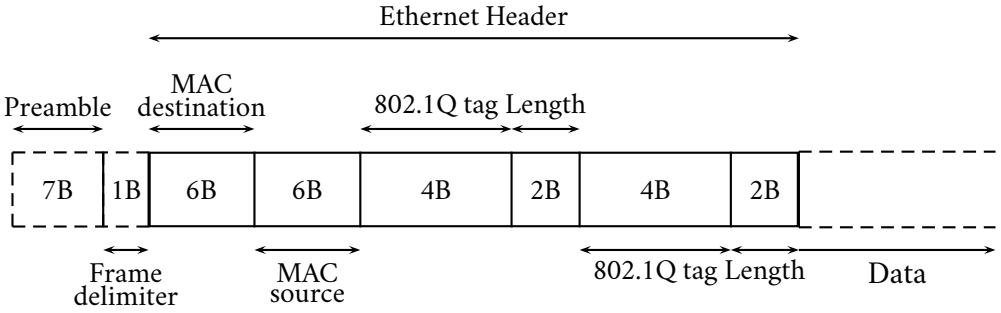


Figure 14.5: MC management with Ethernet 802.1Q-in-Q

First, we add a 802.1Q tag dedicated to MC management. Then, we add a second 802.1Q tag used for VLAN attribution and priority management.In the case where we want to manage several different criticality levels (at least 3), we will add a second IEEE 802.1Q tag in Ethernet header. We will use all 4 bits (PCP, DEI) to indicate a potential criticality level (see figure 14.6). It means that, we will be able to manage $2^4 = 8$ different criticality levels.
In a way, this process represents an extension of the Ethernet 802.1Q header: we add another 802.1Q to the header for criticality management purposes. This implies redefining the Ethernet header structure inside the switches of the network. It means that all the switches in the network will have to be configured specifically for criticality management, in order to be compliant with this new protocol and to be able to read the MC data contained in a message header. This reconfiguration can be done in 802.1Q and 802.1Q-in-Q compliant switches. It means that we will not have to change the materials composing the topology. This approach is important to integrate in order to satisfy MC integration inside COTS networks.

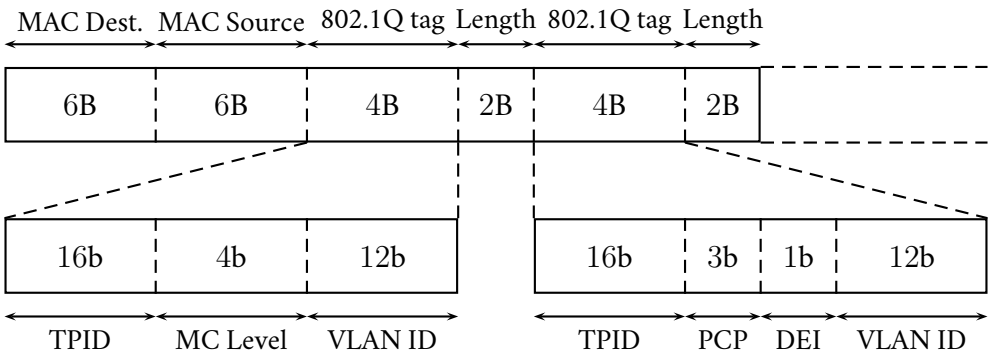


Figure 14.6: 802.1Q-in-Q tags for MC integration inside Ethernet

14.2 Conclusion

Besides the presented solution proposes a solution to integrate MC tagging inside Ethernet networks, we rely on concrete implementations inside benchmarking platforms to provide simulation results.

References

[1] M. Audiard and G. Lautner, *Les tontons flingueurs*. Gaumont, 1963.

[2] B. Pascal, *Pensées*. Guillaume Després, 1669.

[3] Collective, *Oxford dictionary of english*. Oxford University Press, 2006.

[4] G. Buttazzo, *Hard real-time computing systems: Predictable scheduling algorithms and applications*, 3rd. Springer Publishing Company, Incorporated, 2011, ISBN: 1461406757, 9781461406754.

[5] H. Kopetz, *Real-time systems*. Springer, 1998.

[6] N. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *Workshop on real-time operating systems and software*, ser. RTOSS, IEEE, 1991, pp. 133–137.

[7] N. Audsley and A. Burns, "Real-time system scheduling," Tech. Rep., 1990.

[8] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Real time systems symposium*, ser. RTSS, IEEE, 1989, pp. 166–171. DOI: 10.1109/REAL.1989.63567.

[9] M. Kargahi and A. Movaghar, "Non-preemptive earliest-deadline-first scheduling policy: A performance study," in *International symposium on modeling, analysis, and simulation of computer and telecommunication systems*, ser. MASCOTS, IEEE, 2005, pp. 201–208.

[10] B. Guillem, A. Colin, and S. Petters, "Wcet analysis of probabilistic hard real-time systems," in *Real-time systems symposium*, ser. RTSS, Washington, DC, USA: IEEE, 2002, pp. 279–, ISBN: 0-7695-1851-6.

[11] C. Ferdinand and R. Heckmann, "Worst-case execution time prediction by static program analysis," in *International federation for information processing*, ser. IFIP, vol. 156, Springer, 2004.

[12] L. Yue, I. Bate, T. Nolte, and L. Cucu-Grosjean, "A new way about using statistical analysis of worst-case execution times," in *Special issue related to the wip session of the 23rd euromicro conference on real-time systems (ecrts'11)*, vol. 8, ACM, 2011, pp. 11–14.

[13] S. Baruah and A. Burns, "Sustainable scheduling analysis," in *International real-time systems symposium*, 2006, pp. 159–168. DOI: 10.1109/RTSS.2006.47.

[14] C. Ferdinand, R. Heckmann, L. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm, *Reliable and precise wcet determination for a real-life processor*. Springer, 2001, pp. 469–485.

[15] A. Colin and S. Peters, "Experimental evaluation of code properties for wcet analysis," in *Real time systems symposium*, IEEE, 2003, pp. 190–199.

[16] J. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Real time systems symposium*, ser. RTSS, IEEE, 1990, pp. 201–209.

[17] F. Eisenbrand, N. Hähnle, M. Niemeier, M. Skutella, J. Verschae, and A. Wiese, "Scheduling periodic tasks in a hard real-time environment," in *International colloquium on automata, languages, and programming*, ser. ICALP, Springer, 2010, pp. 299–311.

[18] B. Sprunt, "Aperiodic task scheduling for real-time systems," PhD thesis, 1990.

[19] J. L. B. Sprunt L. Sha, "Scheduling sporadic and aperiodic events in a hard real-time system," Software Engineering Institute, Tech. Rep., 1989.

[20] Plato, *Republic*. Plato, 354BC.

[21] M. T. J. Jasperneite P. Neumann and K. Watson, "Deterministic real-time communication with switched ethernet," in *Workshop on factory communication systems*, ser. WFCS, IEEE, 2002, pp. 11–18.

[22] E. Ziouva and T. Antonakopoulos, "Csma/ca performance under high traffic conditions: Throughput and delay analysis," *Computer communcations*, no. 25, pp. 313–321, 2022.

[23] A. Colvin, "A. with collision avoidance," *Computer communications*, no. 6, pp. 227–235, 1983.

[24] G. Leen, D. Heffernan, and A. Dunne, "Digital networks in the automotive vehicle," *Computing and control engineering journal, vol. 10*, vol. 10, no. 6, pp. 257–66, 1999.

[25] J. Meditch and C.-T. Lea, "Stability and optimization of the csma and csma/cd channels," *Ieee transactions on communications, vol. 31*, vol. 31, no. 6, pp. 763–774, 1983.

[26] *Ieee standard for management information base (mib) definitions for ethernet*, 501 Hoes Lane 3rd Floor Piscataway NJ 08855: IEEE, 2013.

[27] G. Ungerboeck, "10gbase-t: 10gbit/s ethernet over copper," Broadcom, Tech. Rep., 2013.

[28] D. D. Chowdhury, *High speed lan technology handbook*. Springer, 2000.

[29] IEEE, "Guidelines for 48-bit global identifier (eui-48)," IEEE, Tech. Rep., 2015.

[30] *The universal administration of lan mac addresses began with the xerox corporation administering block identifiers (block ids) for ethernet addresses.* 501 Hoes Lane 3rd Floor Piscataway NJ 08855, 2002.

[31] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Ire*, vol. 1, pp. 228–235, 1961.

[32] C. Systems, "Design best practices for latency optimization," Cisco, Tech. Rep., 2007.

[33] A. Mifdaoui, "Spécification et validation d'un réseau de communication de type ethernet commuté pour systèmes avioniques militaires de nouvelles générations," PhD thesis, 2007.

[34] *Arinc 664*, 501 Hoes Lane 3rd Floor Piscataway NJ 08855: ACCE, 2002-2008.

[35] *Arinc 429*, 501 Hoes Lane 3rd Floor Piscataway NJ 08855: ACCE, 1983.

[36] I. Land and J. Elliott, "Architecting arinc 664, part 7 (afdx) solutions," Tech. Rep., 2009.

[37] G. Garner, F. Feng, K. D. Hollander, and H. Jeong, "Ieee 802.1 avb and its application in carrier-grade ethernet," *Eee communications magazine*, vol. 45, pp. 126–134, 2007.

[38] L. H. J. Imtiaz J. Jaspernelte, "A performance study of ethernet audio video bridging (avb) for industrial real-time communication," in *Conference on emerging technologies and factory automation*, ser. ETFA, IEEE, 2009, pp. 1–8.

[39] *Audio-video bridging*, 501 Hoes Lane 3rd Floor Piscataway NJ 08855, 2011.

[40] J. J. J. Park B. Cheoun, "Worst-case analysis of ethernet avb in automotive system," in *International conference on information and automation*, ser. ICIAfS, IEEE, 2015, pp. 1696–1699.

[41] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the ieee*, vol. 91, pp. 112–126, 2003.

[42] S. Choi, "Cyclic polling-based dynamic bandwidth allocation for differentiated classes of service in ethernet passive optical networks," *Photonic network communications*, vol. 7, pp. 87–96, 2004.

[43] M. Plankensteiner, "Ttethernet: A powerful network solution for all purposes," Tech. Rep., 2010.

[44] E. Gavrilut, D. Tämas-Selicean, and P. Pop), "Fault-tolerant topology selection for ttethernet networks," in *Safety and reliability of complex engineered systems conference*, ser. ESREL, Citeseer, 2015.

[45] R. Marau, L. Almeida, and P. Pedreiras, "Enhancing real-time communication over cots ethernet switches," in *Workshop on factory communication systems*, ser. WFCS, vol. 6, Citeseer, 2006, pp. 295–302.

[46] *Ieee standard for local and metropolitan area networks: Media access control (mac) bridges*, 501 Hoes Lane 3rd Floor Piscataway NJ 08855, 2011.

[47] D. L. Mills, "Internet time synchronization: The network time protocol," *Ieee transactions on communications*, vol. 39, pp. 1482–1493, 1991.

[48] *Precision time protocol (ptp) version 2 specification*, 501 Hoes Lane 3rd Floor Piscataway NJ 08855, 2008.

[49] C. S. Ltd, "Implementing ieee1588v2 for use in the mobile blackhaul," IEEE, Tech. Rep.

[50] M. Ouellette, K. Ji, S. Liu, and H. Li, "Using ieee 1588 and boundary clocks for clock synchronization in telecom networks," *Ieee communications magazine*, vol. 49, no. 2, pp. 164–171, 2011.

[51] Confucius, *The analects*. c. 4th century BC, 475BC.

[52] S. Martin, P. Minet, and L. George, "End-to-end response time with fixed priority scheduling: Trajectory approach versus holistic approach," *International journal of communication systems*, vol. 18, no. 1, pp. 37–56, 2005.

[53] K. Tindell, A. Burns, and A. Wellings, "Calculating controller area network (can) message response times," *Control engineering practice*, vol. 3, no. 8, pp. 1163–1169, 1995.

[54] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocessing and microprogramming - parallel processing in embedded real-time systems*, vol. 40, pp. 117–134, 1994.

[55] F. Ridouard, J. Scharbarg, and C. Fraboul, "Stochastic network calculus for end-to-end delay evaluation of avionics multi-hop virtual links," *Ifac proceedings volumes*, vol. 40, no. 22, pp. 383–390, 2007.

[56] J. L. Boudec and P. Thiran, *Network calculus, a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2012, vol. 2050.

[57] M. Boyer and C. Fraboul, "Tightening end to end delay upper bound for afdx network calculus with rate latency fifo servers using network calculus," in *Workshop on factory communication systems*, ser. WFCS, IEEE, 2008, pp. 11–20.

[58] A. J. A. Bouillard, "Worst-case delay bounds with fixed priorities using network calculus," in *International conference on performance evaluation methodologies and tools*, ser. ICST, ICST, 2011, pp. 381–390.

[59] J. Schmitt, F. A. Zdarsky, and M. Fidler, "Delay bounds under arbitrary multiplexing: When network calculus leaves you in the lurch...," in *Conference on computer communications*, ser. INFOCOM, IEEE, 2008, pp. 1669–1677.

[60] S. Martin, "Maîtrise de la dimension temporelle de la qualité de service dans les réseaux," PhD thesis, Université Paris XII, 2004.

[61] S. Martin and P. Minet, "Schedulability analysis of flows scheduled with FIFO: Application to the expedited forwarding class," in *International parallel and distributed processing symposium*, ser. IPDPS, Rhodes Island, Greece: IEEE, 2006, 8–pp.

[62] G. Kemayo, F. Ridouard, H. Bauer, and P. Richard, "Optimistic problem in the trajectory approach in FIFO context," in *Conference on emerging technologies and factory automation*, ser. ETFA, IEEE, 2013, pp. 1–8.

[63] O. Cros, X. Li, and L. George, "The trajectory approach for afdx fifo networks revisited and corrected," in *International conference on embedded and real-time computing systems and applications*, ser. RTCSA, IEEE, 2014, pp. 1–10.

[64] X. Li, J. Scharbarg, and C. Fraboul, "Analysis of the pessimism of the trajectory approach for upper bounding end-to-end delay of sporadic flows sharing a switched ethernet network," in *Real-time networks symposium*, ser. RTSS, IEEE, 2011, pp. 149–158.

[65] H. Bauer, J.-L. Scharbarg, and C. Fraboul, "Improving the worst case delay analysis of an afdx network using and optimized trajectory approach," *Ieee transactions on industrial informatics*, vol. 6, pp. 521–533, 2010.

[66] G. Kemayo, F. Ridouard, H. Bauer, and P. Richard, "Optimism due to serialization in the trajectory approach for switched ethernet networks," in *Junior workshop on real-time computing*, ser. JRWRTC, IEEE, 2013, pp. 13–16.

[67] I. Asimov, *I, robot*. Gnome Press, 1950.

[68] M. Kwiatkowska, G. Norman, and D. Parker, "Verification of probabilistic real-time systems," in *Ecole temps reel*, ser. ETR, ACM, 2011, pp. 585–591.

[69] M Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos: A model-checking tool for real-time systems," in *Workshop on formal techniques in real-time and fault tolerant system*, ser. FTRTFT-FORMATS, Springer, 1998, pp. 298–302.

[70] D. D. R. Alur C. Courcoubetis, "Model-checking for probabilistic real-time systems," in *International colloquium on automata, languages, and programming*, ser. ICALP, Springer, 1991, pp. 115–126.

[71] J. Nutaro, *Building software for simulation: Theory and algorithms, with applications in c++*. 2010.

[72] R. Kaiser and S. Wagner, "Evolution of the pikeos microkernel," in *International workshop on microkernels for embedded systems*, ser. MIKES, NICTA, 2007, p. 50.

[73] B. Michelson, "Event-driven architecture overview," Patricia Seybold Group, Tech. Rep., 2006.

[74] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: A flexible real time scheduling framework," in *The special interest group on ada*, ser. SIGAda, vol. 24, ACM, 2004, pp. 1–8.

[75] O. M. Group, "Uml profile for marte: Modeling and analysis of real-time embedded systems," Tech. Rep., 2011.

[76] E. Maes, "Validation de systèmes temps-réel et embarqué à partir d'un modèle marte," Tech. Rep., 2007.

[77] I. Chuang, "Quantum algorithm for distributed clock synchronization," *Physical review letters*, vol. 85, 2006.

[78] P. H. Feiler, B. A. Lewis, and S. Vestal, "The sae architecture analysis and design language (aadl) a standard for engineering performance critical systems," in *International conference on computer aided control system design, international conference on control applications, international symposium on intelligent control*, ser. CACSD, IEEE, 2006.

[79] F. Singhoff, J. Legrand, and L. Nana, "Aadl resource requirements analysis with cheddar," Tech. Rep., 2005.

[80] G. Lasnier, B. Zalila, L. Pautet, and J. Hugues, "Ocarina: An environment for aadl models analysis and automatic code generation for high integrity applications," in *Ada-europe international conference*, ser. Ada-Europe, Springer, 2009, pp. 237–250.

[81] R. Urunuela, A. Déplanche, and Y. Trinquet, "Storm: A simulation tool for real-time multiprocessor scheduling evaluation," in *Conference on emerging technologies and factory automation*, ser. ETFA, IEEE, 2009, pp. 1–8.

[82] J. Drake, M. Harbour, J. Gutierrez, P. L. Martinez, J. Medina, and J. Palencia, "Mast: Modeling and analysis suite for real time applications," Tech. Rep., 2014.

[83] P. Munk, "Visualization of scheduling in real-time embedded systems," PhD thesis, Institute of Software Technology, University of Stuttgart, 2012.

[84] M. Chéramy, P. Hladik, and A. Déplanche, "Simso: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *International workshop on analysis tools and methodologies for embedded and real-time systems*, ser. WATERS, Euromicro, 2014.

[85] K. Muller and T. Vignaux, "Simpy: Simulating systems in python," *Onlamp. com python devcenter*, 2003.

[86] P. Courbin and L. George, "Fortas: Framework for real-time analysis and simulation," in *Workshop on analysis tools and methodologies for embedded and real-time systems*, ser. WATERS, 2011, pp. 21–26.

[87] P. Pettersson and K. Larsen, "Uppaal2k," *Bulletin of the european association for theoretical computer science*, vol. 70, pp. 40–44, 2000.

[88] Y. Chandarli, M. Qamhieh, F. Fauberteau, and D. Masson, "Yartiss: A generic, modular and energy-aware scheduling simulator for real-time multiprocessor systems," in *Laboratoire de l'institut gaspard monge*, ser. LIGM, 2014.

[89] S. Altmeyer, N. Navet, and L. Fejoz, "Using cpal to model and validate the timing behaviour of embedded systems," in *Workshop on analysis tools for embedded and real-time systems*, ser. WATERS, Euromicro, 2015.

[90] K. Diemer, P. Axer, and R. Ernst, "Compositional performance analysis in python with pycpa," in *Workshop on analysis tools for embedded and real-time systems*, ser. WATERS, Euromicro, 2012.

[91] A. Hamann, R. Henia, R. Racu, M. Jersak, K. Richter, and R. Ernst, "Symta/s-symbolic timing analysis for systems," in *Wip proc. euromicro conference on real-time systems 2004 (ecrts'04)*, Citeseer, 2004, pp. 17–20.

[92] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi, "Times: A tool for schedulability analysis and code generation of real-time systems," in *International workshop on formal modeling and analysis of timed systems)*, ser. FORMATS, Springer, 2003, pp. 60–72.

[93] M. Köksal, *A survey of network simulators supporting wireless networks*. Graduate School of Natural and Applied Sciences, 2008.

[94] M. Karl, "A comparison of the architecture of network simulators ns-2 and tossim," Institut für Parallele und Verteilte Systeme, Tech. Rep., 2005.

[95] K. Harju and S. Korventausta, "Network simulation and protocol implementation using network simulator 2," TTKK, Tech. Rep., 2001.

[96] J. Ousterhout, "Tcl: An embeddable command language," EECS Department, University of California, Berkeley, Tech. Rep. UCB/CSD-89-541, 1989.

[97] D. Mahrenholz and S. Ivanov, "Real-time network emulation with ns-2," in *International symposium on distributed simulation and real-time applications*, ser. DS-RT, IEEE, 2004, pp. 29–36.

[98] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *International conference on simulation tools and techniques for communications, networks and systems and workshops*, ser. SIMUTOOLS, ICST, 2008, p. 60.

[99] A. Varga, "The omnet++ discrete event simulation system," in *European simulation multiconference*, ser. ESM, vol. 9, SN, 2001, p. 65.

[100] ——, "Omnet++ user guide," OpenSim, Tech. Rep., 2014.

[101] J. Outerhout, "Tcl and the tk toolkit," Tech. Rep., 1994.

[102] T. Steinbach, H. Kenfack, F. Korf, and T. Schmidt, "An extension of the omnet++ inet framework for simulating real-time ethernet with high accuracy," in *4th international icst conference on simulation tools and techniques*, ser. SIMUTOOLS, ICST, 2011, pp. 375–382.

[103] D. Pediaditakis, S. H. Mohajerani, and A. Boulis, "Poster, the difference of accurate simulation in wireless sensor networks," in *European conference on wireless sensor networks*, ser. EWSN, ACM, 2007.

[104] C. Morgan, *Discrete-event system simulation*. Pearson, 2012.

[105] A. Varga, "Parameterized topologies for simulation programs," in *Communication networks and distributed systems*, ser. CNDS, 1998, pp. 11–14.

[106] X. Chang, "Network simulations with opnet," in *Conference on winter simulation: Simulation—a bridge to the future*, ser. WSC, ACM, 1999, pp. 307–314.

[107] T. M. R. Coelho M. Szczepanski and G. Fohler, "A web based monitoring tool for afdx networks," in *Workshop on analysis tools for embedded and real-time systems*, ser. WATERS, Euromicro, 2015.

[108] D. Crockford, "The application/json media type for javascript object notation (json)," Network Working Group, Tech. Rep., 2006.

[109] G. Murray, "Asynchronous javascript technology and xml (ajax) with the java platform," Tech. Rep., 2005.

[110] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *Workshop on analyzing tools and methodologies for embedded and real-time systems*, ser. WATERS, Euromicro, 2010, pp. 6–11.

[111] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems*, vol. 30, no. 1-2, pp. 129–154, 2005.

[112] J. Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance evaluation*, vol. 2, pp. 237–250, 1982.

[113] H. Hoang, M. Jonsson, U. Hagstrom, and A. Kallerdahl, "Switched real-time ethernet with earliest deadline first scheduling protocols and traffic handling," in *Parallel and distributed processing symposium*, ser. IPDPS, IEEE, 2001, 6–pp.

[114] S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance.," in *Real time systems symposium*, ser. RTSS, IEEE, 2007, pp. 239–243.

[115] S. Baruah and S. Vestal, "Schedulability analysis of sporadic tasks with multiple criticality specifications," in *Euromicro conference on real-time systems*, ser. ECRTS, Euromicro, 2008, pp. 147–155.

[116] B. Huber, C. E. Salloum, and R. Obermaisser, "A resource management framework for mixed-criticality embedded systems," in *Conference of industrial electronics society*, ser. IECON, IEEE, 2008, pp. 2425–2431.

[117] A. Burns and R. Davis, *Mixed criticality systems: A review*. Department of Computer Science, University of York, 2013, vol. Tech. Rep.

[118] S. Baruah, "Mixed criticality schedulability analysis is highly intractable," *2009-08-25. http://www.cs.unc.edu/baruah/pubs.shtml*, 2009.

[119] S. Baruah, H. Li, and L. Stougie, "Towards the design of certifiable mixed-criticality systems," in *Real-time and embedded technology and applications symposium*, ser. RTAS, IEEE, 2010, pp. 13–22.

[120] J. Real and A. Crespo, "Mode change protocols for real-time systems: A survey and a new proposal," *Real-time syst.*, vol. 26, no. 2, pp. 161–197, Mar. 2004, ISSN: 0922-6443. DOI: 10.1023/B:TIME.0000016129.97430.c6.

[121] M. Abuteir and R. Obermaisser, "Mixed-criticality systems based on time-triggered ethernet with multiple ring topologies," in *International symposium on industrial embedded systems*, ser. SIES, IEEE, 2014, pp. 170–178.

[122] M. Jakovljevic, "Synchronous/asynchronous ethernet networking for mixed criticality systems," in *Digital avionics systems conference*, ser. DACS, IEEE, 2009, 1–E.

[123] A. Burns, J. Harbin, and L. S. Indrusiak, "A wormhole noc protocol for mixed criticality systems," in *Real-time systems symposium*, ser. RTSS, IEEE, 2014, pp. 184–195.

[124] O.Cros, F. Fauberteau, X.Li, and L.George, "Mixed-criticality over switched ethernet networks," *Ada user journal*, vol. 35, pp. 138–143, 2014.

[125] S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, N. Megow, and L. Stougie, "Scheduling real-time mixed-criticality jobs," *Ieee transactions on computers , vol. 61*, pp. 1140–1152, 2012.

[126] X. L. O. Cros L. George, "A protocol for mixed-criticality management in switched ethernet networks," in *Workshop on mixed criticality - real time systems symposium*, ser. WMC-RTSS, IEEE, 2015.

[127] K. Gibran, *Sand and foam*. Will Jonson, 1926.

[128] X. Défago, A. Schiper, and P. Urbán, "Total order broadcast and multicast algorithms: Taxonomy and survey," *Acm comput. surv.*, vol. 36, no. 4, pp. 372–421, Dec. 2004, ISSN: 0360-0300. DOI: 10.1145/1041680.1041682.

[129] L.George and P.Minet, "A fifo worst case analysis for a hard real-time distributed problem with consistency constraints," in *International conference on distributed computing systems*, ser. ICDCS, IEEE, 1997, pp. 441–448.

[130] B. Hu, K. Huang, G. Chen, L. Cheng, and A. Knoll, "Adaptive runtime shaping for mixed-criticality systems," in *International conference on embedded software*, ser. EMSOFT, 2015, pp. 11–20.

[131] J. W. von Goethe, *Elective affinities, book i*. J. G. Cottaische Buchhandlung, Berlin, 1809.

[132] O. Cros, F. Fauberteau, L. George, and X. Li, "Simulating real-time and embedded networks scheduling scenarios with artemis," in *Workshop on analysis tools for embedded and real-time systems*, ser. WATERS, 2014, p. 43.

[133] L. G. O. Cros, "Mixed-criticality management of networked real-time systems with artemis simulator," in *Workshop on analysis tools for embedded and real-time systems*, ser. WATERS, Euromicro, 2015.

[134] C. Grannell, *Css and html web design*. friendsof, 2007.

[135] M. Mokhonoana, *The confessions of a misfit: Reasons why i suck so much*. Two Way, 2011.

[136] M. de Cervantes, *El ingenioso hidalgo don quixote de la mancha*. Francisco de Robles, 1615.

[137] G. E. O. Cros L. George, "Dynamic criticality management with artemis," in *Workshop on analysis tools for embedded and real-time systems*, ser. WATERS, Euromicro, 2016.

[138] H. P. Lovecraft, *The defence remains open !* Collected Essays, 1921.

[139] D. Eddings, *Pawn of prophecy*. Ballantine Books, 1982.

[140] F. G. T. Humernbrum and S. Gorlatch, "Using software-defined networking for real-time internet applications," in *International multiconference of engineers and computer scientists (imecs'14)*, vol. I, 2014, pp. 150–155.

[141] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of open vswitch," in *12th usenix symposium on networked systems design and implementation (nsdi'15)*, Oakland, CA: USENIX Association, May 2015, pp. 117–130, ISBN: 978-1-931971-218.

[142] J. Tolkien, *The lord of the rings: The fellowship of the ring*. George Allen & Unwin, 1954.

[143] A. Nothomb, *Stupeur et tremblements*. Albin Michel, 1999.

[144] C. Systems, "Ieee 802.1q-in-q vlan tag termination," Tech. Rep., 2009.

Acronyms

AADL Architecture Analysis and Design Language. 81, 82, 91

AFDX Avionics Full DupleX switched Ethernet. 34, 40–43, 46–49, 74, 88, 89, 93, 94, 122, 179, 183, 192, 208, 212, 231, 250

API Application Programmable Interface. 93, 94, 201, 233

ARTEMIS Another Real-Time Engine for Message Integration Simulation. 12, 14, 16, 164, 182, 183, 193, 194, 197, 200–212, 214–218, 220–222, 224, 226, 227, 230–235

ATM Asynchronous Transfer Mode. 40

AVB Audio-Video Bridging. 35, 43–46, 49, 88, 89

BCET Best Case Excution Time. 96

. 217, 218, 220

BMCA Best Master-Clock Algorithm. 51

CAN Controller Area Network. 35, 88, 89, 105, 110, 122, 179, 208, 212

COTS Commercial Off The Shelf. 14, 34–36, 40, 41, 43, 49, 171, 175, 176, 178, 179, 230, 233, 234

CPU Computing Unit. 83, 96

CRIT Critical. 186

CSMA Carrier-Sense Multiple Access. 46

DEI Drop Eligible Indicator. 39, 40

EDF Earliest Deadline First. 26

FIFO First In First Out. 26, 39, 65, 67, 104, 204

Fortas Framework fOr Real-Time Analysis and Simulation. 84, 86, 87, 98

FP Fixed Priority. 26, 29, 65, 204

FTT-SE Flexible Time Triggered Switched Ethernet. 49

GPS Global Positioning System. 22, 109, 110, 119

GUI Graphical User Interface. 82, 84, 87, 90–93, 95, 96, 201–203, 205, 209–212, 217, 235

HI High. 107, 115, 116, 120, 121, 125, 127, 135, 136, 140, 141, 145, 150, 155–167, 175–177, 216, 222–225, 251

HTTP Hyper Text Transfer Protocol. 209

IoT Internet of Things. 22, 168

IRIT Institut de Recherche en Informatique de Toulouse. 235

LAN Local Area Network. 35, 36, 38, 91, 93

LED Light-Emitting Diode. 111

LO Low. 107, 115, 116, 120, 121, 125, 127, 135, 136, 140, 141, 145, 150, 155–163, 165–168, 175–177, 194, 216, 217, 222–225, 251, 252

MAC Media Access Control. 28, 29, 39, 42

MARTE Modeling and Analysis of Real-Time and Embedded systems. 75, 76, 81, 84, 87

MAST Modeling and Analyzing Suite for Real-Time Applications. 83–85, 87

MC Mixed Criticality. 14–16, 34–36, 82, 87, 95, 101, 103, 106–114, 116, 117, 119–123, 125, 126, 128–130, 132, 134, 135, 137, 151, 152, 154–156, 158, 161, 162, 164, 165, 167, 168, 171–179, 181, 184, 194, 195, 212, 214–217, 220–222, 224–227, 230–235, 252, 254

MISS Mission-Critical. 186

NED Network DEscription. 90, 91

NoC Network-on-Chip. 212

NONC Non-Critical. 186

NS Network Simulator. 88, 89, 91–93

NTP Network Time Protocol. 50

OSI Open Systems Interconnection. 77, 87, 92, 171, 176, 178

PCP Priority Code Point. 39, 40

PDCA Plan Do Check Act. 93

PTP Precision Time Protocol. 12, 14, 35, 49–55, 123, 124, 131–134, 140, 154, 171–176, 179, 231, 234

PTP-ETE PTP-End To End. 52–54, 171

PTP-P2P PTP-Peer To Peer. 53–56, 171

QoS Quality Of Service. 14, 43, 44, 46, 49, 76, 106, 154–156, 161, 163, 165–168, 207, 230, 233, 236

RM Rate-Monotonic. 26

RPI Requesting Port Identifier. 55

RT Real-Time. 14–16, 20–30, 34–36, 38–41, 44–46, 49, 50, 54, 55, 58, 60, 62, 72, 74–89, 92, 93, 95–101, 103, 104, 106–114, 116, 119, 122, 123, 125, 151, 152, 154, 155, 164, 168, 171, 177, 183, 194, 200, 201, 203, 212, 214, 215, 227, 230–236

SAFE Safety-Critical. 186

SCC Switch-Criticality Call. 124, 126–133, 135–137, 139–141, 143, 154, 159, 172, 174, 175, 177, 215, 221

SDN Software-Defined Networks. 233

STORM Simulation Tool for Real-time Multiprocessor Simulation. 82–85, 87, 93

TCI Tag Control Information. 39, 40

TDMA Time Division Multiple Access. 47, 48

TTEthernet Time-Triggered Ethernet. 34, 35, 47–49, 179, 208, 231

UDP User Datagram Protocol. 171, 172, 176

UML Unified Modeling Language. 75

VEHI Vehicle-Critical. 186

VLAN Virtual Local Area Network. 39–42, 47, 48, 124, 131–133, 171, 172, 177, 178

WAN Wide Area Network. 35

WCET Worst Case Execution Time. 26, 27, 29, 77, 80, 81, 83, 86, 96–99, 184

WCAT Worst Case Analyzing Time. 31, 38, 44, 48, 50, 59, 63–65, 67, 68, 108, 111–116, 124, 125, 128–130, 132, 133, 135, 137, 139–143, 146–148, 155, 157, 158, 160, 162, 166, 176, 177, 184–186, 191, 194, 195, 202, 204, 211, 215–218, 220–223, 230

[type=acronym]

List of Figures

List of Algorithms

Gestion de la criticité mixte dans les réseaux temps-réel embarqués et application à Ethernet commuté

Résumé

Dans les domaines industriels tels que les transports publics, le spatial ou l'aéronautique, toutes les commandes du système sont centralisées via l'ordinateur de bord du véhicule. L'exécution de commandes est alors assurée par la transmission de messages via un réseau embarqué. Lors de situations critiques (détection de chocs, phase de décollage, etc...) la transmission des messages critiques (liés aux fonctions cruciales pour l'appareil telles que les commandes de pilotage, la mesure de vitesse, etc...) doit être garantie. L'objectif de cette thèse est de proposer une solution garantissant dans un même réseau la gestion des messages de différentes criticités. Ainsi, durant les phases critiques, les messages de plus haute criticité seront transmis dans un temps borné et ne seront pas retardés par la transmission des informations moins critiques. Cette thèse traite le problème en trois phases : proposer des protocoles de gestion de la criticité dans un réseau et valider ce protocole par un modèle de calcul de délai, montrer la fiabilité de ces protocoles via la conception d'un outil de simulation et enfin proposer une implémentation dans les réseaux Ethernet.

Mixed criticality management into real-time and embedded network architectures: application to switched Ethernet networks

Abstract

In safety-critical industrial domains such as public transports, spacecraft or avionics, all commands have been centralized in the system's dashboard. During critical situations (crash detection, landing phase, etc...) crucial functions (piloting functions, speed measurement, etc...) transmission has to be guaranteed. The purpose of this thesis is to propose criticality management protocols in order to mix messages of different criticalities inside the same infrastructure. Then, non-critical functions transmission will not delay critical messages transmission. This thesis is splitted in three different parts : proposing criticality management protocols for embedded networks and validating critical messages end-to-end transmission time, validating the reliability of these protocols through the conception of a simulation tool and eventually proposing an implementation of these protocols inside Ethernet networks.