ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE CONCENTRATION RÉSEAUX DE TÉLÉCOMMUNICATIONS
M. Sc. A.

PAR
Dhafer ABIDI

ÉTUDE ET SIMULATION DU PROTOCOLE TTETHERNET SUR UN SOUS-SYSTÈME DE GESTION DE VOLS ET ADAPTATION DE LA PLANIFICATION DES TÂCHES À DES FINS DE SIMULATION

MONTRÉAL, LE 20 MAI 2015

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. Abdelouahed Gherbi, directeur de mémoire
Département de génie logiciel et de TI à l'École de technologie supérieure

M. Alain Abran, président du jury
Département de génie logiciel et de TI à l'École de technologie supérieure

M. Alain April, membre du jury
Département de génie logiciel et de TI à l'École de technologie supérieure

M. Sardaouna Hamadou, examinateur externe
École polytechnique de Montréal

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 5 MAI 2015

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

C’est avec le plus grand plaisir que je réserve cette page en signe de gratitude et de reconnaissance à tous ceux qui m’ont aidé de près ou de loin à la réalisation de ce travail.

Mes remerciements s’adressent à mon directeur de mémoire Abdelouahed Gherbi qui m’a permis de mener à terme ce travail par ses conseils et encouragements.

De même je tiens à remercier les membres de jury pour l’honneur qu’ils m’ont fait en acceptant de juger mon travail

ÉTUDE ET SIMULATION DU PROTOCOLE TTETHERNET SUR UN SOUS-SYSTÈME DE GESTION DE VOLS ET ADAPTATION DE LA PLANIFICATION DES TÂCHES À DES FINS DE SIMULATION

Dhafer ABIDI

RÉSUMÉ

TTEthernet est une technologie réseau déterministe qui permet d'apporter des améliorations à la qualité de services de la couche 2 d'Ethernet. Les composants implémentant ces services enrichissent les fonctionnalités d'Ethernet avec une synchronisation distribuée tolérante aux fautes, un partitionnement temporel robuste de la bande passante et une communication synchrone avec une latence fixe et une très faible gigue.

Les services de TTEthernet permettent de faciliter la conception de systèmes distribués robustes, moins complexes et évolutifs capables de tolérer des défaillances multiples.

La simulation constitue, de nos jours, une étape incontournable dans le processus de conception de systèmes critiques et représente un support précieux pour la validation et l'évaluation des performances.

CoRE4INET est un projet regroupant l'ensemble des modèles de simulation de TTEthernet disponible actuellement. Il se base sur l'extension des modèles du framework INET d'OMNeT++.

Notre objectif est d'étudier et de simuler le protocole TTEthernet sur un sous-système de gestion de vols (FMS).

L'idée est d'utiliser CoRE4INET pour concevoir le modèle de simulation du système cible.

Le problème est que CoRE4INET n'offre pas un outil de planification de tâches pour le réseau TTEthernet.

Pour remédier à ce problème on propose une adaptation, pour des fins de simulation, d'une approche de planification de tâches basée sur la spécification formelle des contraintes réseau.

L'utilisation du solveur Yices a permis la traduction de l'ensemble des spécifications formelles en un programme exécutable générant le plan de transmission souhaité.

Une étude de cas nous a permis, à la fin, d'évaluer l'impact de l'agencement des instants d'envoi des trames TT sur les performances de chaque type de trafic du système.

Mots Clés : TTEthernet, simulation OMNeT++, CoRE4INET, FMS, planification des tâches

STUDY AND SIMULATION OF TTETHERNET PROTOCOL ON A FLIGHT MANAGEMENT SUBSYSTEM AND ADAPTATION OF SCHEDULING TASKS FOR SIMULATION PURPOSES

Dhafer ABIDI

ABSTRACT

TTEthernet is a deterministic network technology that makes enhancements to Layer 2 Quality-of-Service (QoS) for Ethernet. The components that implement its services enrich the Ethernet functionality with distributed fault-tolerant synchronization, robust temporal partitioning bandwidth and synchronous communication with fixed latency and low jitter.

TTEthernet services can facilitate the design of scalable, robust, less complex distributed systems and architectures tolerant to faults.

Simulation is nowadays an essential step in critical systems design process and represents a valuable support for validation and performance evaluation.

CoRE4INET is a project bringing together all TTEthernet simulation models currently available. It is based on the extension of models of OMNeT ++ INET framework.

Our objective is to study and simulate the TTEthernet protocol on a flight management sub-system (FMS).

The idea is to use CoRE4INET to design the simulation model of the target system.

The problem is that CoRE4INET does not offer a task scheduling tool for TTEthernet network.

To overcome this problem we propose an adaptation for simulation purposes of a task scheduling approach based on formal specification of network constraints.

The use of Yices solver allowed the translation of the formal specification into an executable program to generate the desired transmission plan.

A case study allowed us at the end to assess the impact of the arrangement of Time-Triggered frames offsets on the performance of each type of the system traffic.

Keywords: TTEthernet, OMNeT++ simulation, CoRE4INET, FMS, scheduling tasks

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

CoRE4INET	Communication over Realtime Ethernet for INET Framework
AS6802	TTEthernet Standard
AW	Acceptance Window
BE	Best Effort
CM	Compression Master
FIFO	First In First Out
FMS	Flight Management System
GHz	Gigahertz (10^9 Hz)
Hz	Hertz
IEEE 802.3	Ethernet Standard
Mbps	Megabits per second (10^6 bps)
PCF	Protocol Control Frames
QoS	Quality of Service
RC	Rate Constrained
SC	Synchronization Client
send_pit	send point in time
SM	Synchronization Master
TT	Time-Triggered
TTEthernet	Time-Triggered Ethernet

INTRODUCTION

La technologie Ethernet a régné sur le monde des réseaux informatiques pendant plus de trente ans. Cependant, l'émergence de nouvelles applications critiques avec des contraintes temporelles strictes et des exigences en matière de tolérance aux fautes, sûreté et sécurité ont poussé plusieurs compagnies à s'activer à la recherche de nouvelles approches répondant à ces nouveaux types demandes. Depuis, des solutions telles que LXI, AFDX et FlexRay se disputent pour gagner une place sur le marché, ciblant plusieurs domaines. Le problème commun de ces innovations récentes est leur peu d'adaptabilité avec le réseau Ethernet classique, ses composants et ses services. Leur évolutivité est limitée et elles ne sont adaptées qu'à un champ spécifique d'applications. C'est de là que la technologie TTEthernet prend toute son essence.

TTEthernet est une technologie réseau déterministe développée par la compagnie TTTech. Elle a été standardisée par la Society of Automotive Engineers en 2011 sous la norme SAE AS6802 pour l'utilisation en systèmes de contrôle en avionique, automobile et plusieurs autres applications industrielles. Elle s'intègre d'une manière transparente avec les composants du standard IEEE 802.3 déjà établi.

Effectivement, TTEthernet apporte des améliorations à la qualité de services de la couche 2 d'Ethernet. Les composants implémentant ces services enrichissent les fonctionnalités d'Ethernet avec une synchronisation distribuée tolérante aux fautes, un partitionnement temporel robuste de la bande passante et une communication synchrone avec une latence fixe et une très faible gigue. De ce fait, les applications critiques de contrôle et de commande, les programmes de divertissement et d'autres applications standards de réseaux locaux peuvent coexister sans les limites imposées par les autres variantes temps réel d'Ethernet.

Les services de TTEthernet permettent de faciliter la conception de systèmes distribués robustes, moins complexes et évolutifs capables de tolérer des défaillances multiples.

La simulation constitue de nos jours une étape incontournable dans le processus de conception de systèmes critiques et représente un support précieux pour la validation et l'évaluation des performances.

L'élaboration d'une simulation repose sur un ensemble de modèles implémentant les protocoles, les composants et les modules de la technologie cible. Ces modèles sont souvent groupés dans des projets tels que INET et Castalia d' OMNeT++. A l'heure actuelle, les seules modèles associés à la technologie TTEthernet sont fournis par le projet CoRE4INET (Communication over Real time for INET framework). Ce projet se base sur l'extension des modèles du framework INET pour implémenter les fonctions de ses propres éléments.

Notre objectif pour ce mémoire est l'étude et la simulation du protocole TTEthernet sur un sous-système de gestion de vol (FMS).

En se basant sur la spécification SAE AS6802 et la revue de littérature, l'étude de TTEthernet portera sur l'ensemble des principes et algorithmes qui régissent ce protocole. On y abordera et discutera les mécanismes de gestion de flux et les différentes politiques d'intégration possibles ainsi que les algorithmes de synchronisation et de tolérance aux fautes.

Une comparaison du modèle de simulation basé CoRE4INET d'OMNeT++ et le modèle basé OPNET a été faite afin d'explorer l'ensemble des solutions offertes et de mettre le point sur leur degré de praticabilité.

Dans le cadre de la présentation de l'approche de planification statique des messages Time Triggered (TT) évoquée par (Steiner, 2010), on abordera les différentes contraintes qui pèsent sur la communication TTEthernet et le rôle que joue la programmation par contraintes dans la résolution de ces problèmes.

La réalisation de notre simulation du protocole TTEthernet nécessite la conception du sous-système FMS. Cela est fait grâce à l'utilisation des modèles CoRE4INET pour bâtir les blocs fonctionnels tout en respectant le mode de fonctionnement du système. Le problème est que malgré l'importance de la planification des tâches pour le protocole TTEthernet, CoRE4INET se contente de définir les blocs fonctionnels d'un réseau TTEthernet et ne fournit pas un outil de planification. Pour pallier à ce problème, on propose, dans le contexte de cette recherche, une adaptation d'une approche de planification basée sur la spécification formelle des contraintes du réseau TTEthernet. Cette adaptation nous a permis de définir avec précision les instants d'envoi (send_pits) des différentes instances de trames TT répondant à un ensemble d'exigences telles que l'exclusion mutuelle de l'accès aux ressources communes, la synchronisation entre les différents nœuds du réseau, la détermination de la taille des buffers pour les commutateurs et la fixation des délais de bout en bout.

La traduction de la spécification formelle des contraintes en un programme exécutable se fait en concordance avec les paradigmes de la programmation par contraintes en utilisant les solveurs de l'outil autonome Yices. Cet outil nous a permis de vérifier la satisfiabilité de l'ensemble de contraintes qui pèsent sur le réseau TTEthernet pour générer ensuite, si c'est le cas, un modèle possible de solutions.

La grande maniabilité que nous a offert notre adaptation de l'approche de planification en termes gestion des tâches nous a permis d'élaborer une étude de cas qui vise à évaluer l'impact de l'agencement des instants d'envoi des trames TT sur les performances de chaque type de trafic du réseau TTEthernet.

Le mémoire comporte quatre chapitres structurés comme suit :

Le premier chapitre donne, dans un premier temps, un aperçu général du réseau TTEthernet et ses principes de bases. Une présentation plus approfondie a été accordée ensuite aux outils utilisés lors de la réalisation de ce travail, à savoir l'environnement de simulation OMNeT++

et le solveur Yices. Pour chacun de ces outils, on présentera le principe de fonctionnement, l'architecture et les différentes facilités indispensables pour le processus de simulation et de planification des tâches.

Le deuxième chapitre aborde en détail le protocole TTEthernet. On y étudiera et discutera les différents types de trafics et leurs méthodes d'intégration en compétition. On procèdera de même pour les algorithmes de synchronisation et de tolérance aux fautes ainsi que leurs modes de fonctionnement.

Ce chapitre passe également en revue la littérature concernant les deux implémentations de modèles de simulations TTEthernet existants actuellement. On y effectuera une comparaison structurelle et fonctionnelle de chacune de ces deux approches.

On examinera, à la fin la notion de programmation par contraintes et comment peut-on résoudre un problème de planification à l'aide de la spécification formelle des contraintes réseau. On présentera dans ce contexte une approche de planification statique des messages TT évoquée par (Steiner, 2010) et on précisera les différentes contraintes du problème ainsi que leurs spécifications formelles.

Le troisième chapitre montre les différentes étapes de réalisation d'un modèle de simulation TTEthernet sur un sous-système de gestion de vols (FMS). On y décrira le processus de conception, allant de l'analyse fonctionnelle du sous-système FMS jusqu'à la construction de ses différents blocs fonctionnelles à partir des modèles CoRE4INET. On présentera et justifiera l'utilisation de chaque modèle de composant tout en précisant les classes implémentant leurs fonctions.

Ce chapitre aborde également les exigences de la simulation pour la planification des trames TT. On y précisera les différentes adaptations apportées à l'approche mentionnée par (Steiner, 2010) afin de générer un plan de transmission répondant aux besoins de la simulation. Les résultats de cette adaptation seront présentés à la fin de ce chapitre.

Le dernier chapitre présente une série d'expérimentations qui visent à valider le comportement du protocole TTEthernet régissant le fonctionnement de notre système du point de vue gestion et distribution de flux ainsi que le processus de correction d'horloge. On y décrira, à la fin une étude de cas, de l'impact de l'agencement contigu et distribué des différents instants d'envoi des trames TT sur les performances de chaque type de trafic du système.

On conclura ce mémoire par une discussion des résultats et une présentation des limites et perspectives de notre travail.

CHAPITRE 1

TTETHERNET ET OUTILS DE SIMULATION

1.1 Introduction

TTEthernet est actuellement une technologie réseau très sollicitée par plusieurs secteurs de pointes tels que l'industrie automobile et l'aérospatiale vu la réponse qu'elle apporte aux défis auxquels sont confrontées les nouvelles applications en matière de bande passante, de fiabilité et de services temps-réels. Cependant, comme toute technologie récente, de grands efforts de modélisation, de vérification formelle et surtout de simulation sont encore requis afin de contrôler et de s'assurer des performances du nouveau protocole.

Ce chapitre comporte trois sections :

La première section définit le protocole TTEthernet et recense, d'une façon sommaire, ses différentes caractéristiques. Une étude plus approfondie des différents aspects du protocole ainsi que les approches de modélisation seront abordés au chapitre suivant.

La deuxième partie se consacre à étudier le simulateur qu'on va se servir pour modéliser le réseau TTEthernet, qui est OMNeT++. On y aborde ses différents concepts, son environnement de travail, ainsi que les outils aidant à bâtir un modèle de simulation et de l'exploiter.

Une comparaison avec deux de ses principaux concurrents (NS et OPNET) sera abordée à la fin de cette section, afin de mettre en valeur l'importance d'OMNeT++.

La dernière section s'intéresse à l'outil de programmation par contraintes *Yices* qui va servir par la suite afin d'élaborer un plan global des différentes tâches pour un réseau TTEthernet.

1.2 Introduction à TTEthernet

TTEthernet ou Time-Triggered Ethernet est une extension temps-réel du protocole Ethernet standard. Elle a été spécifiée par TTTech en collaboration avec Honeywell et standardisée ensuite sous le nom SAE AS6802 (SAE, 2011) par la société SAE (Society of Automotive Engineers). Elle est conçue pour supporter la transmission de flux de données de différentes criticités temporelles au sein du même réseau.

TTEthernet fournit trois services de communication reliés à trois classes de trafic ayant des exigences temporelles différentes :

Le service à déclenchement temporel associé à la classe de trafic Time-Triggered (TT) est celui de plus haute criticité : il permet de garantir une transmission avec une faible gigue et une latence limité. Ceci est réalisable grâce au maintien d'une synchronisation globale entre les différents nœuds du réseau et au suivi, de même, d'un plan de communication globale permettant d'éviter les conflits entre les différents messages TT.

Un deuxième service de criticité moyenne est assuré par TTEthernet. Il concerne le trafic Rate Constrained (RC). Ce service garantit une bande passante de bout en bout et, contrairement au trafic TT, il est asynchrone. Sa latence maximale peut être calculée hors ligne et elle est souvent beaucoup plus élevée que la latence du trafic TT. Cela revient essentiellement au temps d'attente qu'encourent les messages RC dans les files d'attentes des commutateurs à travers le réseau.

Finalement, le trafic le moins critique est transmis selon l'approche best effort d'Ethernet, sans garantie de délai ou de réception. Cette classe de trafic est nommée également Best Effort (BE).

TTEthernet fonctionne parfaitement avec des terminaux Ethernet standards qui n'implémentent pas le protocole Time-Triggered. Cependant ces derniers restent désynchronisés et communiquent seulement via un trafic de type Best-Effort.

1.2.1 Structure d'un réseau TTEthernet

Un réseau TTEthernet est composé de terminaux et de commutateurs. Les terminaux se connectent aux commutateurs avec des liaisons bidirectionnelles. Les commutateurs peuvent se connecter entre eux dans une configuration à sauts-multiples.
La structure d'un réseau TTEthernet est conforme avec celle d'Ethernet commuté et suit par conséquent une topologie en étoile.

La prise en charge du mécanisme de tolérance aux fautes impose au réseau TTEthernet une organisation disjointe et redondante à concurrence de trois canaux de communication pour les implémentations actuelles. La tolérance aux fautes est requise dans les systèmes à haute-sûreté tels que les avions civils et les vaisseaux spatiaux qui doivent rester fonctionnels même avec la présence d'échecs.

La figure 1.1 montre un réseau TTEthernet redondant qui tolère deux équipements défaillants sans dégradation de services. Ce réseau est composé de trois canaux redondants et de quatre terminaux. Chaque canal est formé par un seul commutateur et les quatre terminaux.

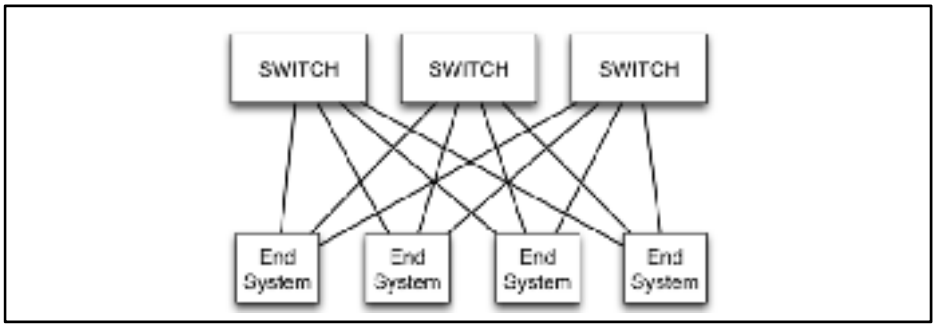

Figure 1.1 Réseau TTEthernet avec double tolérance aux fautes
Tirée de Dutertre et al. (2012)

1.2.2 Les protocoles de TTEthernet

Afin de pouvoir assurer une communication Time-Triggered, TTEthernet doit accorder une grande importance à établir et maintenir une base de temps commune entres les différents équipements du réseau tout en tenant compte des éventuels échecs qui peuvent surgir aux niveaux des commutateurs et des terminaux.

La base de temps commune est une condition nécessaire pour garantir le caractère déterministe au trafic TT et de même, satisfaire l'exigence d'une communication à faible gigue et une latence fixe. La synchronisation étroite va permettre aussi aux différents nœuds de suivre un plan de communication commun, calculé au préalable.

Pour réaliser ces objectifs, TTEthernet recourt, d'abord, au protocole *startup*. Ceci permet d'établir une synchronisation suite à une mise en marche ou un redémarrage. Un protocole de synchronisation d'horloges maintient la synchronisation en corrigeant périodiquement les déviations possibles d'horloges.

Enfin, un protocole de détection de cliques permet de diagnostiquer les services et de détecter les composants défaillants. Ceci permet d'isoler les éléments défaillants hors du processus de synchronisation et de résoudre certaines perturbations transitoires du réseau.

Dans tous ces protocoles, chaque nœud est assigné à un rôle particulier :

- Synchronization Masters (SM)
- Compression Masters (CM)
- Synchronization Clients (SC)

Typiquement, les SMs sont des terminaux, les CMs sont des commutateurs, alors que les SCs peuvent être, indifféremment, des commutateurs ou des terminaux.

Les SMs assurent la mise en marche du réseau et le maintien de sa synchronisation : initialement, ils débutent le protocole startup et une fois que le réseau est stabilisé, ils déclenchent périodiquement la synchronisation d'horloges afin de préserver la base de temps commune atteinte au préalable grâce au protocole startup.

Le démarrage de tous les protocoles est marqué par l'envoi de messages de contrôle PCFs. Ce sont des messages de haute priorité transportant des informations de synchronisation d'un ou plusieurs SMs vers les CMs. Chaque CM collecte les PCFs de tous les SMs dont il est connecté, il les filtre et effectue leur traitement. Un nouveau message PCF est créé et sera renvoyé par la suite à tous les nœuds du réseau.

Les SCs ont un rôle passif aux startups et synchronisation d'horloges. Ils se contentent d'écouter la communication et de se synchroniser avec le reste du réseau à la réception d'un PCF valide.

1.3 Environnement de simulation OMNeT++

Les avancées technologiques ont permis aux systèmes informatiques en général, et aux réseaux informatiques en particulier, d'évoluer et de devenir de plus en plus complexes.

Des outils de contrôle, de validation et d'analyse de performance sont alors requis. On en distingue les outils analytiques et les méthodes expérimentales, mais surtout, les simulateurs réseaux, incontournables de nos jours pour le développement d'architectures de communication et des protocoles réseaux.

1.3.1 Introduction à OMNeT++

OMNeT++ opère selon le mode de modélisation à évènements discrets (Klaus, Mesut et James, 2010) où les opérations au sein d'un système sont assimilées à une succession d'évènements maintenus par le simulateur à l'aide d'une file d'attente et triés selon leurs

temps d'exécution. Chaque évènement s'exécute à un instant particulier provoquant un changement de l'état du système. La simulation progresse alors en exécutant les évènements de la file d'attente et profite de la stabilité du système entre deux évènements consécutifs afin de pouvoir sauter directement d'un évènement à l'autre.

Il est important aussi de noter qu'OMNeT++ est un logiciel libre, il dispose d'une licence publique à des fins académiques. Sa vocation première est de créer un outil fiable et compétitif aux simulateurs spécialisés, orientés-recherche tels que NS2 et aussi au simulateur commercial OPNET dont la licence est très onéreuse.

OMNeT++ n'offre pas des modèles de simulation prêts-à-utiliser. Il fournit cependant, un environnement puissant alimenté de tous les outils nécessaires allant du noyau de simulation, les bibliothèques, les éditeurs graphiques et les outils d'analyse permettant de créer ces composants de simulation.

Des modèles de simulation sont alors développés par plusieurs personnes et groupes de recherche et que, indépendamment d'OMNeT++, chacun suit son propre cycle de développement. Ils sont souvent groupés sous forme de paquetage(s), qu'on appelle framework. Parmi eux, on peut citer essentiellement INET framework qui englobe une large panoplie de modèles implémentant divers protocoles tels que les protocoles Internet (TCP, UDP, IPv4, IPv6, …), les protocoles de la couche liaison (Ethernet, PPP, IEEE 802.11, …) et aussi Castalia et MiXiM frameworks dédiés aux réseaux mobiles et aux réseaux fixes sans files.

Ces frameworks sont en constante évolution et gagnent de plus en plus de fiabilité et de richesse, ce qui a permis à OMNeT++ avec le temps de devenir un outil de choix chez plusieurs groupes de recherche universitaires, des institutions de recherche à but non lucratif, mais aussi chez plusieurs compagnies comme IBM, Intel, Cisco, Thales et Broadcom (Varga et Hornig, 2008).

1.3.2 Structure d'un modèle OMNeT++

Un modèle OMNeT++ consiste à un ensemble de modules imbriqués d'une façon hiérarchique et communiquant entre eux à l'aide de messages (Varga et Hornig, 2008).

Le *module système* représente le niveau le plus haut de la hiérarchie. Il contient des sous-modules qui peuvent à leur tour contenir d'autres sous-modules. Le nombre de niveaux de la hiérarchie est illimité.

Les modules qui contiennent d'autres sous-modules s'appellent *modules composés*. Le niveau le plus bas de la hiérarchie comprend les modules dits simples. Ces derniers constituent les composants actifs du modèle ; ils contiennent ses algorithmes et s'implémentent par l'utilisateur à l'aide du langage C++.

Les modules simples et composés sont des instances de type *module.* Ils sont définis par l'utilisateur lors de la description du modèle afin de servir comme composants pour d'autres types de modules plus complexes. On obtient au final le module système qui représente en fait le module réseau et englobe tous les sous-modules sans avoir de connexions vers l'extérieur (voir figure 1.2).

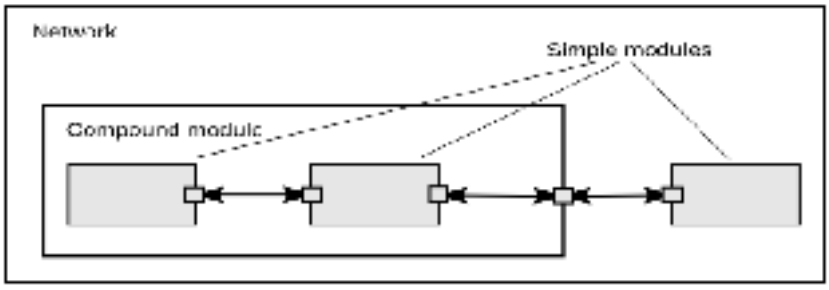

Figure 1.2 Hiérarchie d'un modèle OMNeT++
Tirée de Varga et Hornig (2008)

Quand un type de module est utilisé comme block de construction, il n'y a pas de distinction entre un module simple ou composé. Cela permet à l'utilisateur de diviser d'une manière

transparente un module en plusieurs modules simples pour obtenir un module composé ou au contraire, ré-implémenter les fonctionnalités d'un module composé pour obtenir un seul module simple sans affecter les utilisateurs du type de module.

Dans un modèle OMNeT++, les modules communiquent à l'aide de messages qui en plus des attributs d'horodatage peuvent contenir plusieurs autre champs d'informations tels que l'adresse source, l'adresse de destination, …

Les modules simples envoient ces messages généralement à travers des ports (connus en OMNeT++ sous le nom gates), mais aussi directement vers le(s) module(s) de destination et ce, dans le cas d'une connexion ad-hoc par exemple.

Ces ports représentent les interfaces d'Entrée/Sortie. Un port d'entrée et un autre de sortie peuvent être liés via une *connexion* qui se crée à l'intérieur d'un seul niveau d'hiérarchie de modules, c'est-à-dire une connexion entre deux sous-modules d'un module composé ou entre un sous-module et un module qui le compose. Les connexions à travers différents niveaux de hiérarchies ne sont pas permises car ils constituent un facteur gênant pour la réutilisation de composants dans un modèle.

Tout en respectant la structure hiérarchique de son modèle, les messages passent à travers une série de connexions entre modules simples. Cela se fait d'une manière transparente des modules composés.

Plusieurs propriétés peuvent être attribuées à une connexion, telles que le délai de propagation, le débit et le taux d'erreurs. Des types de connections appelés *channels* peuvent être alors spécifiés afin de fixer ces propriétés et faciliter leurs réutilisations pour d'autres connexions.

Enfin, on peut associer à chaque module des paramètres qui aideront par la suite à construire sa topologie, en spécifiant le nombre de nœuds par exemple. Ils sont aussi utiles pour approvisionner le code C++ implémentant les modules simples en entrées.

Ces paramètres peuvent avoir un type numérique, chaine de caractères ou xml. Des valeurs par défaut peuvent leurs être attribués et seront utilisées s'ils ne subiraient pas d'affectations ultérieurement.

De plus, un paramètre peut avoir le qualificatif *volatile* ce qui provoque la relecture de l'expression dans laquelle il figure à chaque fois il est lu. Cela est important, pour la simulation, dans le cas où il modélise une distribution numérique aléatoire.

1.3.3 Le langage de description de réseau (NED)

La topologie d'un modèle d'un modèle OMNeT++ est décrite à l'aide du langage NED (Network Description) (Klaus, Mesut et James, 2010). Il permet à l'utilisateur de déclarer des modules simples, de les connecter et de les assembler dans des modules composés. Il spécifie encore au simulateur le module composé final à simuler en lui attribuant le mot clé *network*.

NED est un langage déclaratif. Il se base, comme présenté précédemment, sur une structure hiérarchique, ce qui permet conjointement avec son fonctionnement orienté-composants d'alléger le degré de complexité des modules implémentés et de favoriser le déploiement de plusieurs bibliothèques comme INET et MiXiM.

Il dispose également de plusieurs autres caractéristiques qui renforcent sa flexibilité et le rendent très adaptable aux grands projets (Varga et Hornig, 2008). On peut en citer :

- **Héritage :** permet de sous-classer des modules et des canaux d'où la possibilité d'y ajouter, au besoin, d'autres membres (paramètres, ports et sous-modules). Il permet aussi

d'ajuster la structure ou le comportement de modules dérivés en changeant leurs paramètres existants.

- **Interfaces :** parfois on a besoin de déterminer le type concret du module ou du canal via un paramètre lors de la configuration du réseau. Les interfaces permettent de réserver les espaces nécessaires en code, là où un module ou un canal doit être utilisé. Ces types concrets doivent, évidemment, implémenter les interfaces à substituer.
- **Paquetages :** l'implémentation de grands projets nécessite souvent de trier les fichiers NED dans différents répertoires. NED supporte une structure arborescente proche de celle du langage Java. Le noyau de simulation est capable de localiser la racine de(s) paquetage(s) grâce à la variable NEDPATH (similaire à CLASSPATH en Java). Cela permet essentiellement de mieux organiser les fichiers NED et surtout réduire les conflits de noms entre les différents modèles.
- **Métadonnées :** des nouvelles propriétés peuvent être attachées, sous forme de métadonnées, à presque tous les attributs du langage NED (modules, canaux, ports, paramètres, paquetages, ...). Celles-ci, ne sont pas utilisées directement par le noyau de simulation, mais peuvent apporter des informations supplémentaires à l'environnement d'exécution et aux modules lors de la simulation ; on les utilise par exemple pour spécifier l'unité de mesure à certains paramètres (exemple : un paramètre delay post-fixé par la propriété @unit(us) doit être exprimé en microsecondes). On en trouve aussi dans les modules d'animation et de représentation graphique (spécification des icônes, des infobulles, …) et dans les outils d'édition et d'invite de commandes où ils sont utilisées comme moyens d'assistance.

De plus, il est important de mentionner que le langage NED possède une représentation XML équivalente ce qui permet de le convertir en fichier XML et inversement sans perte de données. Ceci va permettre d'automatiser le traitement de ces fichiers par des langages tiers, dans le sens où on peut désormais extraire facilement les informations pertinentes, restructurer, transformer un fichier NED et aussi le générer à partir des informations sauvegardées en bases de données.

Il est à noter aussi que les fichiers NED peuvent être édités graphiquement ou en mode texte de l'IDE OMNeT++.

Finalement, NED définit seulement la structure d'un modèle sans se soucier de son comportement et laisse un sous-ensemble de ses paramètres ouvert. Le comportement sera fixé, ensuite, à l'aide d'une implémentation C++ des modules simples. Quant aux paramètres, ils recevront leurs valeurs à partir d'un *fichier ini* dit de configuration.

1.3.4 Programmation des modules simples

OMNeT++ offre un environnement de développement intégré (IDE) avec tous les outils nécessaires pour écrire, exécuter et déboguer un programme (Klaus, Mesut et James, 2010).

Le comportement de chaque module simple est implémenté à l'aide d'une classe C++. Pour le faire, l'utilisateur doit dériver la classe de base *cSimpleModule*, redéfinir ses fonctions virtuelles et enfin, enregistrer la nouvelle classe via le macro *Define_Module()*.

Les interactions entre les différents modules se font souvent par le biais de messages ; ceux-ci sont utilisés pour livrer des informations utiles entre les couches applicatives et aussi, comme un déclencheur d'évènements vu que le noyau de simulation ne distingue pas entre message et évènement.

Le traitement des messages se fait à l'intérieur de la fonction *handleMessage(cMessage *msg)* des classes implémentant les modules simples. Ensuite chaque module peut se décider, soit de renvoyer immédiatement le message traité via la fonction *send(cMessage *msg)*, soit de retarder l'émission pour un certain temps, en s'envoyant un auto-message à l'aide de la fonction *scheduleAt(simtime_t time, cMessage *msg)*.

Les attributs de la classe de base *cMessage* lui permettent de sauvegarder plusieurs informations, à savoir le nom, la longueur, le type, ainsi que d'autres champs véhiculant des

informations supplémentaires telles que le temps d'arrivée, le port de destination et le temps le plus récent d'envoi.

Il est possible aussi d'ajouter des nouveaux attributs à un message en dérivant sa classe de base. Cela se fait automatiquement grâce à l'outil *opp-msgc* de l'OMNeT++ qui génère les fichiers C++ nécessaires à partir d'un fichier message (*.msg*).

Un autre aspect fondamental d'OMNeT++ est la simulation des piles protocolaires réseaux. En effet, les couches d'un réseau sont souvent implémentées comme des modules qui échangent des messages grâce à plusieurs mécanismes : d'une part, grâce aux opérations d'encapsulation/décapsulation implémentées par la classe cMessage et d'autre part, grâce à l'attachement d'un champ d'informations auxiliaires dans un objet nommé *Control Info*. Il sera par la suite enlevé et traité à la réception du message à la couche de destination. Exemple, Quand un datagramme IP descend vers la couche Ethernet, le champ Control Info qui lui est attaché, peut comprendre l'adresse MAC de destination. Dans la direction inverse, ce champ peut contenir l'adresse IP ou la connexion TCP pour les couches supérieures.

Parmi les principales fonctions virtuelles qu'on a besoin de redéfinir lors de l'implémentation d'un modèle OMNeT++ figurent les fonctions d'initialisation et de finalisation. C'est dans la première que se déroule la plupart des opérations d'initialisation, notamment des données membres avec support au mécanisme d'initialisation multi-étapes : cela se fait lorsque le code d'initialisation d'un module dépend d'autres modules déjà initialisés. Dans ce cas, le processus d'initialisation s'exécute progressivement, sur plusieurs étapes. Cette solution est plus propre que l'approche traditionnelle d'autres simulateurs tels qu'OPNET et NS, qui consiste à une diffusion globale des évènements d'initialisation au début de la simulation.

La communication des paramètres d'un fichier NED vers le code C++ d'un module se fait à l'intérieur de la fonction d'initialisation à l'aide de la méthode *par()*.

Pour la fonction de finalisation, elle est utilisée principalement pour enregistrer des résultats statistiques scalaires à la fin d'une simulation réussie.

Il est important de mentionner que la communication via des messages n'est pas toujours la solution la plus efficace, spécialement dans le cas de modules fortement couplés. Il serait plus judicieux alors d'utiliser l'appel direct de méthodes, c'est-à-dire l'appel de fonctions publiques d'un module simple. Cela nécessite cependant, deux manipulations supplémentaires : localiser le module appelé et enregistrer la méthode d'appel auprès du noyau de simulation.

Plusieurs méthodes peuvent être utilisées pour trouver le module appelé selon sa position : dans le cas le plus courant où le module appelé se trouve sur le même module composé que celui de l'appelant, les deux méthodes *getParentModule()* et *getSubModule()* de la classe *cModule* sont utilisées pour renvoyer un pointeur vers le module appelé. Le cas contraire, il faut utiliser la méthode *getModuleByPath(const char *path*) de *cSimulation* pour identifier le module appelé par son chemin absolu.

Le pointeur vers le module appelé pointe initialement sur un objet de type *cModule* ou *cSimulation*. La fonction *check_and_cast<>()* se charge de forcer le typage vers le type actuel.

Pour enregistrer la méthode d'appel auprès du noyau de simulation, l'une des deux méthodes publiques *Enter_Method()* ou *Enter_Method_Silent()* doit être utilisée en dessus. Ceci va déterminer si l'appel va s'afficher à l'interface graphique utilisateur (GUI) ou non et aussi pour d'autres raisons de changement temporaire de contexte.

La bibliothèque INET utilise largement ce mode d'appel pour accéder aux modules comme RoutingTable, InterfaceTable et NotificationBoard d'un nœud.

OMNeT++ fournit également un ensemble performant de fonctions de débogage. Les actions et les évènements qui se produisent lors de l'exécution d'un modèle peuvent être affichés sous forme textuelle ou même sous forme d'animation annotée grâce à Tkenv de GUI. De plus, il est possible de surveiller l'évolution en temps réel des différentes variables d'un module à l'aide de la macro *WATCH()*.

Des classes telles que *cOutVector*, *cStdDev*, *cDoubleHistogram* et *cLongHistogram* sont utilisés pour sauvegarder et calculer des statistiques de base.

Finalement, concernant la relation entre les fichiers NED et l'implémentation C++ de modules, OMNeT++ compile les fichiers NED en code C++ puis fait la liaison avec l'exécutable du simulateur. Ceci est profitable pour pouvoir créer dynamiquement des modules composés : leurs sous-modules et leurs connexions internes peuvent être générés automatiquement grâce au code NED compilé.

L'importance de cet aspect réside dans le fait qu'il permet d'élargir le spectre de scénarios de simulation en OMNeT++ pour comprendre des topologies capables de changer en temps d'exécution et par conséquent, pouvoir répondre à des nouvelles questions d'optimisation, en relation avec la structure du réseau étudié.

1.3.5 Bibliothèques

On peut classer les bibliothèques en OMNeT++ en deux catégories : une première regroupant les bibliothèques de modèles, la deuxième englobe les bibliothèques de simulation.

1.3.5.1 Bibliothèques de modèles

Ces bibliothèques ne sont pas incluses initialement à l'environnement OMNeT++, c'est à la charge de l'utilisateur de les installer. Elles sont développées par des groupes indépendants de professionnels et de chercheurs et sont assimilées à des contributions à OMNeT++.

Les bibliothèques de modèles font souvent partie de grands projets multi-domaines appelés frameworks et offrent une large panoplie de modèles de protocoles, de modèles d'applications et de sources de trafics, ainsi que d'autres composants réutilisables et respectant le paradigme modulaire d'OMNeT++.

Chaque framework suit son propre cycle de publication indépendamment d'OMNeT++. Parmi les plus connus d'OMNeT++, on peut citer INET framework et les frameworks de mobilité (MiXiM, Castalia, …)

1.3.5.1.1. INET framework

INET est considéré comme la bibliothèque de modèles standard d'OMNeT++. Elle était basée initialement sur le paquetage IPSuite développé à l'université Karlstuhe puis maintenu par l'équipe d'OMNeT++ en lui ajoutant des correctifs et de nouveaux modèles.

Elle contient actuellement des modèles de protocoles pour la suite TCP/IP (IPv4, IPv6, TCP, SCTP, UDP, …), des modèles de la couche liaison pour les réseaux filaires et sans fils (Ethernet, PPP, IEEE802.11, …), des modèles MPLS avec signalisation RSVP et LDP, un support à la mobilité et plusieurs autres protocoles et composants.

Ses modules sont organisés dans des paquetages qui sont à leurs tours organisés selon les couches du modèle OSI (exemple : inet.applications, inet.transport, …).

Du point de vue architectural, INET respecte le concept modulaire d'OMNeT++ : les protocoles sont représentés par des modules simples dont les interfaces externes sont décrites par des fichiers NED et le comportement est implémenté à l'aide de classes C++. Les nœuds sont construits par composition de plusieurs modules simples.

D'autres modules (qui n'implémentent pas de protocoles) sont utilisés pour assurer des tâches spécifiques au cours de la simulation :

On en trouve côté nœud, le module InterfaceTable qui contient la table des interfaces réseau (eth0, wlan0, …), les tables de routage *RoutingTable* et *RoutingTable6* pour IPv4 et IPv6 respectivement et le module *NotificationBoard* qui facilite la communication entre les différents modules.

Au niveau réseau, on cite le module *FlatNetworkConfigurator* qui sert à attribuer les adresses IP aux différents nœuds et de configurer un routage statique, le module *ScenarioManager* qui contrôle les expériences de simulation et la planification d'évènements et le module *ChannelControl* requis pour les simulations sans fil et permet de garder la trace des nœuds à l'intérieur d'une zone d'interférences avec d'autres nœuds.

En ce qui concerne l'interaction entre ses différents éléments, INET gère la communication entre les différentes couches de protocoles via un processus d'encapsulation/décapsulation avec Control Info comme un objet attaché au message pour véhiculer une information additionnelle à la couche prochaine.

Un mode d'appel direct est suivi pour lier les autres modules, souvent en communication. Cela, est assuré par son module *NotificationBoard* qui joue le rôle d'intermédiaire entre le module où les évènements apparaissent et les modules qui sont intéressés par ces évènements. Son fonctionnement est basé sur le concept publication/abonnement selon lequel les modules peuvent s'abonner à des catégories de changements (exemple : un tableau de routage change d'état, un canal de communication devient libre). Quand l'un des changements se produit, le module hôte (exemple : Table de routage, couche physique) informe le module NotificationBoard, qui à son tour, diffuse l'information vers tous les modules à cette catégorie de changement.

INET constitue aujourd'hui un framework incontournable pour OMNeT++. La richesse de ses modèles et la réutilisabilité de ses composants lui ont garanti un large déploiement chez la communauté OMNeT++ et lui ont permis d'être le socle sur lequel se basent plusieurs extensions telle que CoRE4INET l'extension INET implémentant le protocole TTEthernet.

1.3.5.1.2. Les frameworks de mobilité

On distingue actuellement les deux frameworks : Castalia et MiXiM. Ils sont désignés pour les réseaux mobiles et sans fils (réseaux de capteurs, BAN, ad-hoc, réseaux de véhicules, …) et visent généralement les dispositifs réseaux embarqués à faible puissance.

Ils offrent des modèles pour la propagation des ondes radio, l'estimation d'interférence, la consommation en énergie des émetteurs/récepteurs radios et les protocoles MAC sans fils.

Les chercheurs les utilisent pour tester les protocoles distribués avec des contextes de modèles radio, canaux sans fils et comportement de nœuds réalistes. Le fait qu'ils sont hautement paramétrables leurs permet aussi d'évaluer les caractéristiques d'une grande variété de plateformes pour des applications spécifiques.

1.3.5.2 La bibliothèque de simulation

La bibliothèque de simulation est le constituant de base du noyau de simulation conjointement avec le code qui gère la simulation. Elle offre une vaste collection de classes C++ indispensables pour implémenter les modules simples, notamment celles définissant les différentes parties d'un modèle de composant : modules, canaux, paramètres de modules et autres objets.

La classe *cMessage* représente les évènements et les messages échangés entres les modules alors que les objets qui qui servent comme conteneurs de sauvegarde sont représentés par les classes *cArray* et *cQueue*. Un objet de type *cQueue* peut être configuré pour fonctionner comme une file de priorité.

La bibliothèque contient également une classe d'exploration de topologie qui permet d'extraire dans un graphe la topologie d'un modèle spécifique afin de pouvoir, par la suite, naviguer à travers ses nœuds et y appliquer des algorithmes de routage tel que l'algorithme de plus court chemin DijKstra.

La génération de nombres aléatoires est prise en charge selon plusieurs distributions continues (uniform, exponential, normal, …) et discrètes (binomial, bernoulli, poisson, …), avec la possibilité d'implémenter de nouvelles distributions par l'utilisateur et de les rendre ensuite disponibles à partir des fichiers NED et de configuration.

Enfin, il y existe plusieurs classes qui gèrent la génération de statistiques. Elles varient de simples classes de collection de moyennes et de déviations standards des échantillons aux classes plus sophistiquées et hautement configurables d'estimation de différentes distributions. Des vecteurs de résultats collectionnés en fonction du temps de simulation sont gérés par la classe *cVector*.

1.3.6 Architecture interne

La bibliothèque de modèles de composants ou Model Component Library (voir figure1.3) représente le code compilé des modules simples et composés. Ces derniers sont instanciés au début de l'exécution de la simulation par le noyau de simulation (SIM) afin de créer le modèle concret.

L'environnement de la simulation est formé par les bibliothèques d'interfaces utilisateur (Env, Cmdenv et Tkenv) qui gèrent l'entrée de données, l'affichage des résultats de simulation ainsi que les informations de débogage et contrôle généralement l'exécution de la simulation et l'aspect visualisation/animation.

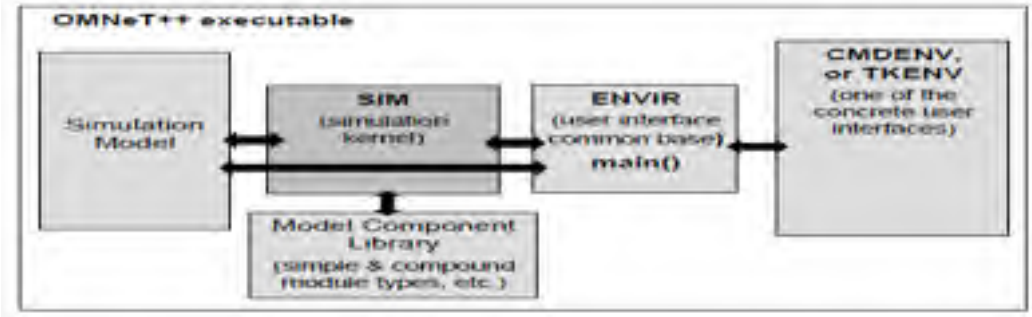

Figure 1.3 Architecture logique d'un programme de simulation OMNeT++
Tirée de Varga et Hornig (2008)

Une simulation OMNeT++ peut être intégrée dans une autre application. Cela est possible grâce à la présence d'une interface générique entre SIM et les bibliothèques d'interfaces utilisateur, à laquelle s'ajoute la séparation physique entre les différentes bibliothèques (SIM, Env, Cmdenv et Tkenv) ce qui permet de remplacer les bibliothèques d'interfaces utilisateur existantes par d'autres parties de l'application hôte. Une base de données, dans ce cas, est nécessaire afin d'alimenter le sous-système OMNeT++ en modèle de topologie.

1.4 Comparaison avec OPNET et NS

Cette partie s'inspire de plusieurs travaux (Klaus, Mesut et James, 2010; Varga et Hornig, 2008; Weingartner, vom Lehn et Wehrle, 2009) de comparaisons de simulateurs réseaux afin de mettre en valeur les caractéristiques d'OMNeT++ par rapport à deux de ses principaux concurrents simulateurs à évènements discrets (NS et OPNET).
Les critères de comparaison sont les suivants :

- **License**
- **Disponibilité du code source (noyau de simulation)** : permet l'intégration et la modification de l'infrastructure de simulation et aussi, un soutien au travail de débogage.
- **Langage de programmation** : influe directement la performance de la simulation en matière de vitesse d'exécution (exemple : une simulation basée C/C++ a une bonne performance comparée à une autre basée Java).
- **Modèle de programmation (fonction de traitement d'évènements/Coroutine)** : influe la structure interne du code et sa consommation mémoire (chaque code de module basé Coroutine s'exécute dans un thread et une pile CPU propre à lui).
- **Débogage et traçabilité** : permet de vérifier que le système travail correctement.
- **Modèles de protocoles disponibles** : c'est l'un des facteurs principaux du succès d'un simulateur et contribue à la diversification de ses domaines d'application.
- **Hiérarchisation et réutilisation de modèles de composants** : permet de réduire la complexité d'un composant en les décomposant en plusieurs sous-modules et allège la programmation en évitant de recopier du code.

- **Définition de la topologie** : cela concerne les moyens de définir la topologie (textuels ou graphiques) et la possibilité de la générer dynamiquement au cours du temps d'exécution.

Tableau 1.1 Comparaison OMNeT++, OPNET et NS

OMNeT++	**OPNET**	**NS**
License		
gratuit	commercial	gratuit
Disponibilité du code source (noyau de simulation)		
disponible	Non disponible	disponible
Langage de programmation		
C++ pour les modules + NED pour la topologie	C pour les modules (la topologie est générée par éditeur graphique et stockée dans un fichier binaire propriétaire)	Script oTcl pour la topologie C++ pour les modules (NS2) Python et C++ (NS3)
Modèle de programmation (fonction de traitement d'évènements/Coroutine)		
Fonction de traitement d'évènements + Coroutine (évènement = message reçu)	Fonction de traitement d'évènements (évènement = message reçu)	Fonction de traitement d'évènements (évènement = message reçu)
Débogage et traçabilité		
Fichier log Animation automatique Fenêtre de sortie par module Inspecteur d'objets	Fichier log Animation automatique Ligne de commande	Fichier log Animation automatique

Modèles de protocoles disponibles		
Riche (TCP/IP, MAC, SCSI, FDDI, MPLS, IEEE802.11, ad-hoc, WSN, TTEthernet, …)	Riche (TCP/IP, ATM, MAC, WiMAX, MPLS, Frame relay, ad-hoc, IEEE802.11 …)	Moins riche Essentiellement TCP/IP
Hiérarchisation et réutilisation de modèles de composants		
Hiérarchique avec un niveau d'imbrication illimité Modèles réutilisables	Hiérarchique avec une restriction au niveau nœud (ne peut pas être imbriqué) Modèles réutilisables	Ne supporte pas la hiérarchisation. (pas d'imbrication possible) Modèles réutilisables
Définition de la topologie		
Textuel (Langage NED) Editeur graphique (Tkenv) Topologie dynamique	Editeur graphique Topologie fixe	oTcl script topologie fixe

1.5 Yices

Yices est un solveur SMT (Satisfiability Modulo Theorie) développé en 2006 par SRI International et, est maintenu continuellement à jour depuis cette date. Il supporte une combinaison riche de théories de premier ordre : théorie de nombres réels, théorie des entiers linéaires, théorie des fonctions non interprétées et des théories de diverses structures de données comme les listes, les tableaux, et les vecteurs de bits. Ces théories sont utilisées fréquemment dans la modélisation logicielle et matérielle.

Yices permet aussi de résoudre les problèmes de pondération MAX-SMT et de construire des modèles. Il est actuellement l'outil de décision par défaut du vérificateur de modèles SAL et de même, il est intégré dans l'assistant de preuve (*theorem prover*) PVS du SRI.

1.5.1 Architecture

Yices est un logiciel basé sur une architecture modulaire (SRI, 2014), ses solveurs de théories sont accessibles via un API ou à partir d'un exécutable Yices. Il est décomposé en trois modules principaux permettant de manipuler les types de données de base d'un API Yices (les termes, les types, les contextes et les modèles). Cette structure est illustrée à travers la figure 2.4 montrant l'architecture haut-niveau de Yices.

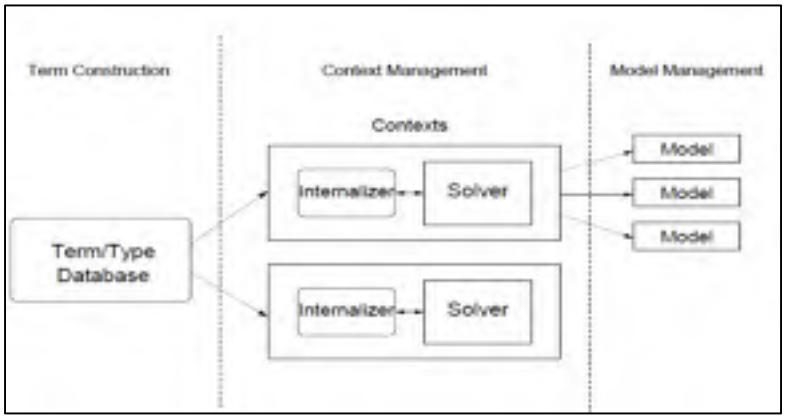

Figure 1.4 Architecture haut niveau de Yices
Tirée de SRI (2014)

Le premier module de base "Term Construction" maintient une base de données globale stockant tous les types et termes. L'API Yices fournit les fonctions nécessaires pour la construction de termes, formules et types dans cette base de données.

Le deuxième module de "Context Management" implémente les opérations applicables sur les contextes. Un contexte est une structure de données centrale sauvegardant un ensemble d'assertions dont on a besoin de vérifier la satisfiabilité. Cela se fait à l'aide de ses deux

composants : le solveur emploie en interne un solveur booléen SAT de satisfiabilité et des procédures de décision afin de déterminer si les formules affirmées (vérifiées par assertions) dans un contexte sont satisfiables. L'initialiseur a pour but de convertir le format utilisé par la base de données de types/termes dans un autre format interne (conjonctive normal) utilisé par le solveur.

L'API Yices inclut les fonctions de gestion permettant la création et l'initialisation de contextes, l'ajout et la suppression des assertions et la vérification de la satisfiabilité des formules affirmées.

Les contextes peuvent être construits et manipulés d'une façon indépendante et ils sont hautement personnalisables du fait que chacun d'entre eux peut être configuré pour supporter une classe spécifique de formules, appeler différentes procédures de prétraitement et de simplification, ainsi que l'utilisation d'un solveur spécifique ou d'une combinaison de solveurs.

Grâce au troisième module "Model Management", un modèle peut être établi au cas où l'ensemble d'assertions d'un contexte sont satisfiables. Un modèle assigne aux symboles des formules une valeur concrète (entier, réel, vecteur de bits constants, …). Il représente aussi un objet indépendant qui peut être interrogé et examiné d'une façon transparente du contexte à partir duquel il a été créé. Le changement ou la suppression de son contexte d'origine ne lui affecte pas.

Enfin, Yices peut être utilisé comme une bibliothèque permettant via ses fonctions de gérer simultanément plusieurs contextes et modèles partageant un ensemble de termes. Ceci est important pour plusieurs mécanismes de contrôle comme celui de contraintes globales exists/forall du solveur SMT.

1.5.2 Les solveurs

Yices permet de sélectionner un solveur spécifique ou une combinaison de solveurs pour gérer un problème donné. Cela a pour effet de pouvoir configurer chaque contexte pour une classe particulière de formules. Par exemple, on peut choisir un solveur spécialisé en arithmétique linéaire ou encore, un solveur général supportant tous les types et structure de données possibles du langage Yices.

La figure suivante résume l'architecture du solveur le plus général en Yices.

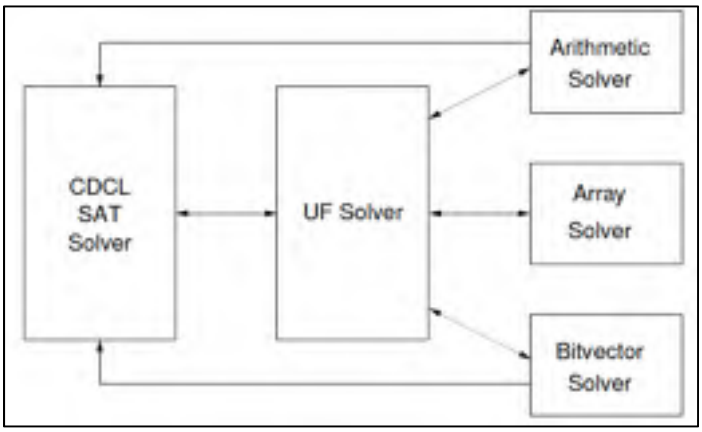

Figure 1.5 Structure d'un solveur général
Tirée de SRI (2014)

Le solveur booléen SAT est un composant indispensable pour les autres solveurs de théories. Il est basé sur l'algorithme Conflict-Driven Clause Learning (CDCL) ou apprentissage de clauses par conflits et fonctionne en couplage avec les solveurs de théories de base, soient le solveur UF (Uninterpreted Functions) dédié aux fonctions non interprétées avec égalité, le solveur arithmétique qui gère l'arithmétique linéaire sur des entiers et réels, le solveur de vecteurs de bits et le solveur de tableaux.

Il est à noter que Yices offre la possibilité d'établir des solveurs spécialisés, plus simples et efficaces, orientés vers des classes de formules spécifiques. Comme exemple, un solveur

arithmétique pur peut être bâti en liant directement le solveur arithmétique avec le solveur CDCL SAT. On procède de même pour obtenir un solveur spécialisé aux problèmes de vecteurs de bits. Toujours dans la même logique, la suppression du solveur arithmétique mène à un solveur spécialisé aux problèmes combinant les fonctions non interprétées, les vecteurs de bits et les tableaux.

La combinaison de plusieurs solveurs de théories passe à travers le solveur UF. Ce dernier assure la coordination entre les différents solveurs et garantie la consistance globale. De ce fait, les solveurs de théories communiquent seulement avec le solveur UF et jamais d'une façon directe. Cela permet de simplifier leur conception et implémentation.

1.5.3 Utilisation de Yices

Yices est distribué soit en version exécutable comme outil autonome, soit sous forme d'une bibliothèque intégrable dans d'autres applications et fournissant les mêmes fonctionnalités qu'un outil autonome. Son utilisation de base se résume par la lecture des formules à partir d'un fichier et la vérification, ensuite, de leurs satisfiabilités. La sortie attendue est un modèle de formules si ces dernières sont satisfiables. Le cas échéant, s'il est impossible de décider de la non-satisfiabilité des formules (cas des formules incluant des quantificateurs par exemple), le résultat de sortie sera "*unknown*".

Yices peut être encore utilisé en mode interactif. Son fonctionnement tourne autour d'un contexte logique interne : un ensemble de déclarations et d'assertions entrées par l'utilisateur dont on peut manipuler grâce à plusieurs commandes permettant principalement d'ajouter ou de supprimer des assertions à l'aide du mécanisme *push/pop* qui permet au contexte de maintenir une pile d'assertions en plusieurs couches. La commande *push* débute une nouvelle couche alors que pop supprime toute les assertions de la dernière. C'est le principe de *backtacking*.

D’autre part, Yices possède son propre langage, mais il accepte aussi les entrées écrites en notation SMT-LIB. L’avantage d’utiliser le langage natif réside du fait qu’il permet d’enrichir l’ensemble des théories de SMT-LIB par d’autres traitant les tuples et les scalaires. Le langage Yices apporte aussi plus de flexibilité que SMT-LIB dans la mesure où il permet un mixage arbitraire des différentes théories et fournit les commandes nécessaires pour accéder à toutes les fonctionnalités de Yices, y compris la construction de modèle, la manipulation de contexte et MAX-SMT.

Le solveur MAX-SMT permet de résoudre des problèmes de pondération, là où on affecte des poids numériques aux assertions afin de trouver le modèle ayant le poids maximal/minimal. Ceci est judicieux dans les problèmes d’optimisation.

1.6 Conclusion

Ce chapitre a servi pour introduire le protocole TTEthernet et ses principes de base. Une étude plus approfondie lui sera accordée dans le prochain chapitre.

Une attention particulière a été donnée à l’environnement de simulation OMNeT++ et l’outil de programmation par contrainte et de vérification formelle Yices qui vont collaborer par la suite pour réaliser le modèle de simulation TTEthernet.

CHAPITRE 2

TTETHERNET : PRINCIPES ET REVUE DE LITTERATURE

2.1 Introduction

En simulation réseau, il est indispensable de bien comprendre le principe de fonctionnement du protocole concerné et ses différentes configurations. C'est dans cette optique qu'on va débuter ce chapitre par une étude des caractéristiques et des différents algorithmes du protocole TTEthernet. Ensuite, on fera une revue de littérature concernant les principaux travaux d'implémentation de modèles de simulation TTEthernet.

Ce travail comporte quatre sections :

La première partie présente les différents types de trafic au sein d'un réseau TTEthernet et discute les différentes méthodes d'intégration de flux synchrones et asynchrones selon plusieurs critères tels que la qualité temps-réel de la transmission, la génération de faux messages et l'efficacité/inefficacité des ressources.

La deuxième section aborde les mécanismes de synchronisation et de tolérance aux fautes pour un réseau TTEthernet, on y présente la fonction de permanence et les algorithmes de synchronisation ainsi que le principe de tolérance de fautes et ses modèles possibles.

La troisième section s'intéresse à la description des différentes implémentations de modèles de simulation existants de TTEthernet, à savoir le modèle basé sur l'extension INET d'OMNeT++ et le modèle d'OPNET.

La dernière partie définit en premier le principe de programmation par contraintes et la spécification formelle de problèmes de planification. Ensuite, elle aborde les travaux qui s'en sont servis pour élaborer un plan de transmission pour le réseau TTEthernet.

Chaque section se termine par une discussion.

2.2 Gestion de trafics TTEthernet

TTEthernet supporte plusieurs types de trafic très différents en termes de leurs caractéristiques et leurs exigences. Leur compétition à s'approprier les ressources du réseau exige d'instaurer un mécanisme de gestion de flux dont le fonctionnement est fondé sur un ordre de priorité attribué à chaque classe de trafic.

2.2.1 Les différents types de trafic

Un réseau TTEthernet peut supporter simultanément jusqu'à trois types de trafic. Ils peuvent être décrits comme suit.

2.2.1.1 Trafic Time-Triggered (TT)

Les messages TT sont utilisés quand des faibles latences et gigues sont requises pour une communication donnée. La transmission du trafic TT nécessite un plan global de transmission, partagé entre les différents nœuds TTEthernet (terminaux et commutateur(s)). Ce plan contient les évènements concernant l'envoi et la réception de messages TTEthernet.

Le routage pour le trafic TT est statique, ce qui le rend complètement déterministe, mais ça nécessite en revanche une configuration hors ligne du système.

D'autre part, le trafic TT utilise un format d'adressage orienté contenu, c'est-à-dire le champ adresse de destination détermine, en plus du nœud cible, le contenu de la trame.
En effet, l'adresse de destination (48 bits) est divisée en deux parties : la première partie de 32 bits sert à identifier le trafic TT. La deuxième partie (16 bits) est nommée CT-ID (Critical Traffic-Identifier), sa valeur est la même pour chaque message dans cycle de

synchronisation. Elle est utilisée pour adresser un message TT selon une planification TTEthernet prédéfinie. La décision du routage en dépend.

2.2.1.2 Trafic Rate Constrained (RC)

Le trafic RC est un trafic asynchrone, utilisé lorsque les données transportées sont moins exigeantes du point de vue du déterminisme et de temps-réel. Cependant, son temps de transmission reste encore borné, puisque, la norme ARINC 664 sur laquelle est basé ce trafic, exige la réservation au préalable d'une bande passante suffisante pour la transmission des messages RC ce qui limite les délais de transmission et les déviations temporelles de ces messages.

Chaque message RC est défini par un identificateur CT-ID qui est l'équivalent logique de VL-ID (Virtual Link- Identifier) pour le standard AFDX.

Pour assurer la bande passante requise pour un message RC, des comptes de l'écart d'allocation de la bande passante (*"Bandwidth Allocation Gap-accounts"* ou *BAG-accounts*) sont définis où BAG-accounts représente le temps minimum entre deux émissions de trames consécutives avec le même CT-ID. L'application qui envoie les messages RC doit respecter les contraintes de BAG-accounts, sinon, le message sera invalidé et rejeté par le commutateur.

2.2.1.3 Trafic Best-Effort (BE)

Le trafic BE correspond au trafic de l'Ethernet classique. Il a la priorité la plus basse parmi les autres classes de trafics et utilise le reste de la bande passante sans garantie d'envoi ou de réception.

2.2.2 Méthodes d'intégration de trafics

L'intégration des trafics synchrones et asynchrones mène souvent à des problèmes de collision. TTEthernet remédie à ce problème par l'attribution à chaque type de trames une priorité. De ce fait, si on veut trier les différents trafics selon l'ordre décroissant de priorité on aura : PCF, TT, RC et BE avec la possibilité d'attribuer plusieurs niveaux de priorités à l'intérieur de la même classe du trafic.

Il existe trois scénarios possibles de conflits :

- **Scénario 1 :** Un conflit entre deux messages de même priorité. Dans ce cas, les messages seront servis selon la méthode d'ordonnancement FIFO.
- **Scénario 2 :** Un conflit se produit lorsqu'un message de haute priorité (H) est en train d'être servi et un message de faible priorité (L) est prêt à être transmis. Dans ce cas, L sera mis en file d'attente.
- **Scénario 3 :** Le troisième conflit est le plus intéressant. Il se pose dans le cas où un message de faible priorité (L) est encours de transmission lorsqu'un nouveau message de haute priorité (H) est prêt à être transmis. Dans ce cas trois approches d'intégration de flux sont possibles : *Preemption, timely bloc et shuffling* (voir figure 2.1). Ces trois approches peuvent être jugées selon plusieurs critères tels que la qualité temps-réel, la génération de faux messages et l'efficacité/inefficacité de l'utilisation des ressources.

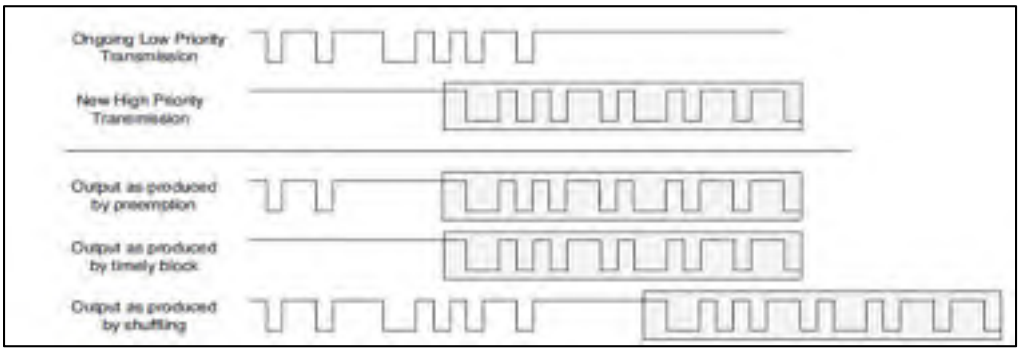

Figure 2.1 Méthodes d'intégration pour les trafics de haute priorité (H) et faible priorité (L)

Tirée de Steiner et al. (2012)

2.2.2.1 Préemption

Pour cette méthode, si un message de moindre priorité (L) est encours de transmission quand un message de haute priorité (H) arrive, la transmission de L sera arrêtée. Le commutateur établira alors un temps minimal de silence au niveau du canal et transmettra ensuite le message H.

Cette méthode représente un bon choix pour les systèmes temps-réels. Elle a pour avantage de garantir au commutateur d'introduire à un message H un temps d'intégration constant et connu à l'avance.

En revanche, les inconvénients de cette méthode se résument dans ces deux points :

- **Génération de faux messages** : la méthode par préemption engendre systématiquement des messages tronqués ce qui lègue au récepteur la tâche de ne pas confondre un message correcte avec un autre tronqué. Cela peut être réalisé par la création d'un modèle de signal qui viole les règles de codage pour la trame de transmission si un message est tronqué ou, simplement, par l'introduction de la taille originale du message à l'intérieur de celui-ci.
- **Inefficacité de l'utilisation de ressources** : un message tronqué sera perdu au niveau du récepteur. Une retransmission de tout le message s'impose, ce qui engendre une perte de bande passante. Cette perte s'accentue si le réseau supporte des nœuds d'Ethernet classique. La solution à envisager dans ce cas est l'ajout d'une fonctionnalité à Ethernet standard permettant de reconstruire les fragments d'un message Ethernet au niveau des récepteurs.

2.2.2.2 Blocage en temps opportun (Timely Block)

La transmission des messages de type TT est configuré à l'avance. Donc, le commutateur connaît à priori quand et sur quel port un message TT va arriver, ainsi que le port vers lequel il doit être adressé.

La méthode Timely Bloc, exploite ces dernières informations. En effet, si le message H est de type TT, le commutateur doit assurer la disponibilité du canal de transmission aux moments où le message TT est planifié. Cela se fait en bloquant les autres messages concurrents.
Cette méthode offre une qualité de service hautement temps-réel aux réseaux TTEthernet. Elle garantit que les ports de sortie d'un commutateur seront libres à l'arrivée du message prioritaire H, ce qui introduit un délai d'intégration quasi-constant pour tous les messages H.

L'inefficacité de l'utilisation de ressources est le principal inconvénient la méthode : dans le cas où les messages de moindre priorité L ont une taille indéfinie, le commutateur/terminal doit avoir une période de blocage correspondant au moins à la taille maximale d'un message L.

La solution consiste à apporter plus de flexibilité au niveau du commutateur/terminal en s'assurant, au cas où la taille du message L est connue, s'il est possible de le transmettre complètement avant l'instant où un message H doit être déclenché.

2.2.2.3 Brassage

Cette technique d'intégration consiste à retarder un message de haute priorité (H), si un autre message moins prioritaire (L) est en cours d'envoi, et ce, jusqu'à la fin de la transmission de ce dernier. On assiste alors à un retard du message H, au pire des cas, pour la durée maximale d'une transmission d'un message L.

L'avantage de cette méthode est l'efficacité de l'utilisation de ressources puisqu'il n'y a ni retransmission des messages tronqués ni perte de bande passante engendrée par le blocage des ports en l'attente des messages H.
Quant aux inconvénients, on peut les résumer dans ces trois points :

- **Qualité temps-réel faible :** si le message H est de type TT, une dégradation de la qualité temps-réel sera observée. La synchronisation globale du TT peut participer à atténuer la gigue. Cependant, il n'y a pas de moyens pour atténuer la durée de transmission réseau.

- **Interférence avec le trafic BE :** le fait qu'un message L peut retarder un autre plus prioritaire (H), en particulier lorsque L est un message de type BE et provenant d'une source incontrôlée, doit être appréhendé avec précaution. L'exemple type est celui d'un passager d'un vol qui se connecte avec son ordinateur portable au réseau de la cabine de l'avion ce qui lui permet d'entrer en interférence avec des données critiques telles que les données d'alarme.
- **Complexité croissante en composabilité et évolutivité pour la gestion du trafic TT :** la méthode de brassage impose un processus de gestion complexe au niveau des ordonnanceurs des messages TT pour faire face à l'augmentation de la durée de transmission et de la gigue. Un travail de planification incrémental ne sera possible que dès le début, lors de la création du plan de communication initiale.

2.2.3 Discussion

L'analyse des différentes méthodes d'intégration de trafics permet de déduire que la méthode avec préemption est la moins adaptée pour un réseau TTEthernet. Cela revient au fait, qu'en plus de la mauvaise exploitation des ressources due aux trames tronquées, les solutions proposées pour remédier à ce problème semblent inappropriées pour des applications temps réel.

En effet, la solution consistant de reprendre la transmission des trames tronquées va accroitre considérablement le temps de livraison des trames moins prioritaires. Par conséquence, une trame RC aurait peu de chance de respecter son échéance s'il arrive qu'elle soit interrompue par une trame TT.

La deuxième solution consistant à implémenter une fonction permettant de reconstruire les fragments des trames tronquées, n'apporte pas d'amélioration significative vue le coût engendré par la gestion de mémoires additionnelles et surtout le coût de traitement et vérification du séquencement qui reste très élevé. En plus, les trames tronquées peuvent être

confondues avec les conséquences d'une faute matérielle ce qui pose un problème additionnel.

Les autres méthodes peuvent être utilisées selon les exigences des applications : *timely block* convient parfaitement avec les applications à hautes exigences temporelles alors que la méthode par brassage présente un compromis entre efficacité d'exploitation de ressources et garantie temporelle.

Enfin, il est à noter que malgré l'importance des méthodes d'intégration de trafics pour un réseau TTEthernet, leurs impacts peuvent être considérablement relaxés avec l'utilisation d'une planification adéquate tenant compte des caractéristiques des applications concurrentes.

2.3 Algorithmes de synchronisation et tolérance aux fautes

La synchronisation au niveau d'un réseau TTEthernet se base sur deux protocoles : le premier est celui de synchronisation d'horloges, le deuxième est nommé protocole Startup/Restart et s'intéresse essentiellement à l'intégration des nouveaux nœuds, la détection de problème de synchronisation et la resynchronisation si nécessaire.

La tolérance aux fautes profite des informations échangées lors du processus de synchronisation pour localiser les nœuds défaillants. Des mécanismes pour renforcer la fiabilité des systèmes peuvent être utilisés.

2.3.1 Synchronisation d'horloges

La synchronisation en TTEthernet se fait grâce à l'échange d'informations entre trois intervenants principales : les Synchronization Masters (SM), les Compression Masters (CM) et les Synchronisation Clients (SC).

La figure ci-dessous montre un exemple de liaisons qui puissent exister entre ces éléments :

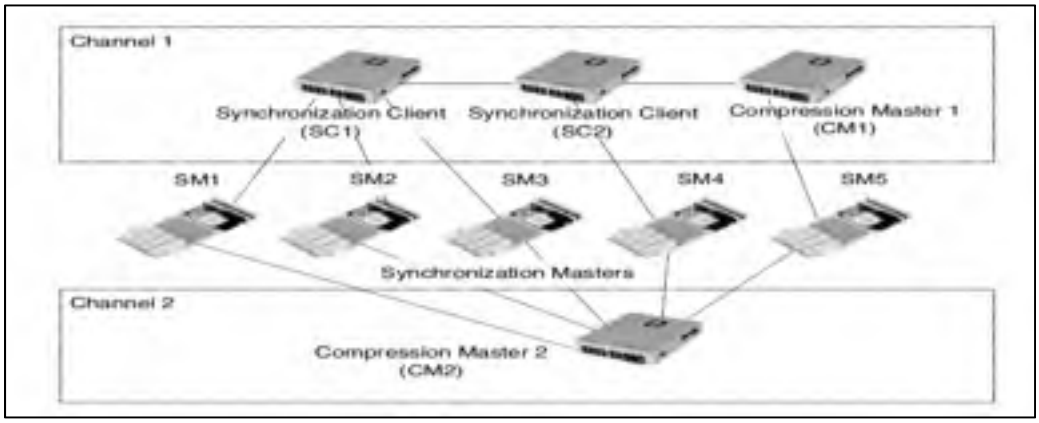

Figure 2.2 Les différents intervenants à la synchronisation TTEthernet
Tirée de Steiner et al. (2012)

L'échange entre ces éléments est valable grâce à une trame Ethernet standard appelée PCF (Protocol Control Frame) servant à transporter les informations pertinentes à la synchronisation entre les différents composants du réseau. Elle est, par ailleurs, issue d'une politique d'intégration non-préemptive, ce qui le rend sujette à des gigues réseau et touche à la qualité de mesure des délais de transmission et par conséquent, à la qualité de synchronisation du système.

Une solution à ce problème est gérée par une fonction dite *fonction de permanence* qui sera décrite dans la section suivante. Une fois la qualité de mesure de délais est assurée par la fonction de permanence, le protocole TTEthernet fait appel à un algorithme de synchronisation d'horloges qui s'étale sur deux étapes et permet d'aboutir à la fin à une synchronisation globale de toutes les horloges du système.

2.3.1.1 La fonction de permanence

Les trames PCFs suivent une politique d'intégration non-préemptive : leurs réceptions ne sont pas planifiées à des points précis dans le temps, puisqu'elles peuvent être générées à la phase d'émission comme aux phases de transmission ou réception.

De ce fait, quand la méthode d'intégration de trafics utilisée est le brassage, la résolution de conflits se fait à travers le recours à des files d'attentes. Cela introduit d'importants délais supplémentaires dont il est indispensable d'atténuer afin d'assurer une synchronisation efficace du système. C'est exactement ce que fait la fonction de permanence. Son principe de fonctionnement est illustré par la figure 2.3.

Pour connaître le temps de bout en bout actuel à partir d'une trame PCF, un mécanisme de mesures doit être mis en œuvre. Il se résume par le fait que chaque composant TTEthernet doit mesurer ses propres délais (délai d'émission, de transmission et de réception). Le résultat de ces mesures (qui est le délai de bout en bout) sera ensuite conservé dans le champ *pcf_transparent_clock* de la charge utile de PCF. Les délais statiques tels que les délais de transmission sur fils peuvent être compensés ensuite par une configuration hors ligne des composants TTEthernet.

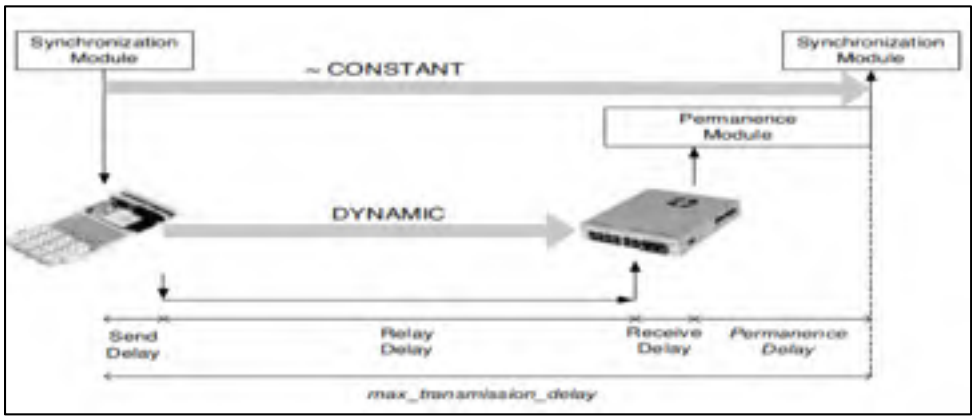

Figure 2.3 La fonction de permanence transforme la gigue réseau en délai de transmission réseau

Tirée de Steiner et al. (2012)

Grâce au champ *pcf_transparent_clock* de la trame PCF, le récepteur est capable de connaître le délai de transmission actuel de la trame reçue. Par la suite, pour combler la variation de délais que peuvent subir différentes trames PCFs, le récepteur applique la fonction de permanence à la trame PCF reçue, en la retardant avec une durée appelée *permanence_delay* (voir Figure 2.3). On dit alors que le récepteur rend permanente la trame PCF.

La variable *permanence_delay* est défini par l'équation suivante :

$$permanence\_delay = max\_transmission\_delay - pcf\_transparent\_clock \tag{2.1}$$

Tirée de Steiner et al. (2012)

max_transmission_delay est un paramètre configuré hors ligne et représente le délai de transmission maximum entre deux nœuds du système TTEthernet (de SM vers CM ou de CM vers SM/CM).

A la fin, on peut conclure que grâce au mécanisme de l'horloge transparente et à la fonction de permanence, un système TTEthernet est capable de transformer le délai dynamique de transmission réseau en un délai constant maximal.

2.3.1.2 La fonction de convergence à deux étapes

La synchronisation des horloges en TTEthernet passe par la fonction de convergence à deux étapes. Pour qu'on puisse la décrire, deux notions doivent être définies (voir Figure 2.4) :

- **Hyper-cycle (Cluster Cycle) :** c'est le temps d'une trame pour un système TTEthernet. Il désigne un cycle complet d'une trame TT pour un plan de transmission donné.
- **Cycle d'intégration (Integration Cycle) :** c'est la période de synchronisation de l'horloge.

Plusieurs Cycles d'intégrations par hyper-cycle contribuent à une haute qualité de précision de mesures et la réintégration efficace des composants qui ont perdu leur synchronisation avec le système.

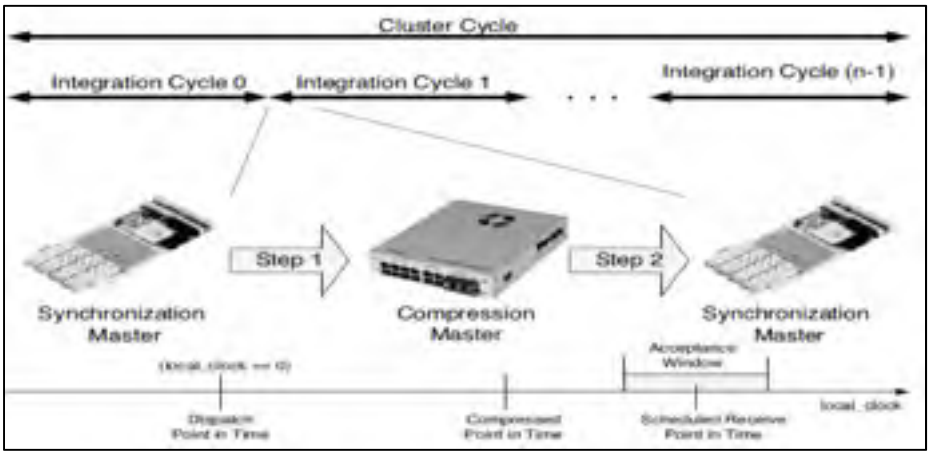

Figure 2.4 La hiérarchie de temporisation dans TTEthernet
Tirée de Steiner et al. (2012)

De plus, un équipement TTEthernet utilise au cours du processus de synchronisation deux variables locales : la première est appelée *local_clock* et est utilisée pour compter cycliquement à l'intérieur d'un cycle d'intégration. La deuxième variable est nommée *local_integration_cycle*. Elle est utilisée pour compter cycliquement le nombre de cycles d'intégration.

Enfin, il est important de noter que l'algorithme de convergence à deux étapes s'exécute au début de chaque cycle d'intégration et opère selon deux étapes de convergence : *Compression Master* et *Synchronization Master.*

2.3.1.2.1. Première étape : Compression Master

Les SMs débutent cette étape par l'envoi en même point de temps (en accord avec leur horloge locale, par exemple quand *local_clock=0*) de leur PCF vers les CMs.

D'autre part, au niveau des récepteurs CMs, les points de réception varient à cause de la variation des oscillateurs des SMs.

Pour compenser ces variations, une fonction de compression est implémentée au niveau des CMs et opère d'une façon asynchrone par rapport au temps globale synchronisé du système. La figure suivante montre les différentes étapes par lesquelles passe une fonction de compression ainsi que ses différents intervenants.

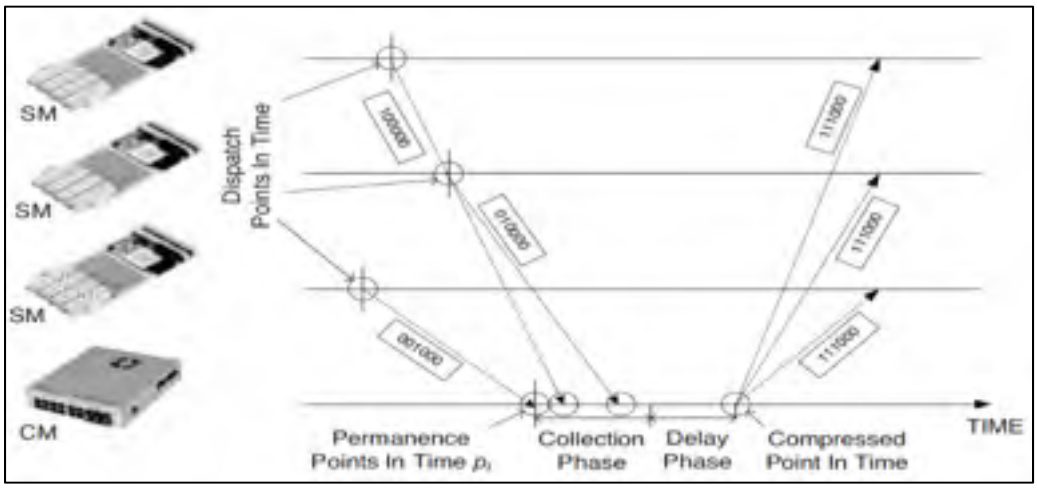

Figure 2.5 Un aperçu du déroulement de la fonction de compression
Tirée de Steiner et al. (2012)

Chaque fonction de compression débute lors de la réception du premier point de permanence de PCFs et continue, par la suite, à recevoir les points de permanence des trames ayant la même valeur *local_integration_cycle* (du même message) pendant un intervalle de temps appelé "*collection phase*".

Les points de permanence des PCFs collectés lors de cette dernière phase seront ensuite exploités pour calculer une valeur de correction "*correction_value*" (SAE, 2011) qui participe à déterminer la durée de la phase d'attente "*Delay Phase*" à laquelle doit attendre

une fonction de compression avant l'émission de sa nouvelle trame PCF appelée "*Compressed PCF*".

De ce fait, la fonction de compression doit attendre pendant un "*delay_phase_duration*" explicité par l'équation suivante :

$$Delay\_phase\_duration = correction\_value + (k+1) * observartion\_window - collection\_phase\_duration. \quad (2.2)$$

Tirée de Steiner et al. (2012)

k est le nombre de SMs défaillants tolérés et *observation_window* est la déviation maximale de deux horloges locales correctes dans un système comme mesurable par une horloge à l'intérieur du réseau.

Enfin, le temps total que prend une fonction de compression à l'intérieur d'un CM, commençant par la réception du premier point de permanence jusqu'à l'émission de sa *compressed PCF* est noté "*compression_master_delay*", il est défini comme suit :

$$\begin{aligned} compression\_master\_delay &= collection\_phase\_duration + delay\_phase\_duration \\ &= (k+1) * observation\_window + correction\_value \end{aligned} \quad (2.3)$$

Tirée de Steiner et al. (2012)

La *compressed PCF* générée à la fin de cette période garde la valeur originale du champ *pcf_integration_cycle* provenant des SMs. En plus, le CM configure le champ *pcf_membership_new* pour qu'il puisse garder la trace des SMs émetteurs des PCFs reçues en phase de collection.

2.3.1.2.2. Deuxième étape : Synchronization Master

La seconde étape de l'algorithme de synchronisation se fait aux niveaux des SMs et SCs.

Dès le début, les points de permanence des *compressed PCFs* sont prédits et configurés hors ligne selon le paramètre suivant :

$$Scheduled\_receive\_pit = dispatch\_pit + 2 * max\_transmission\_delay + compression\_master\_delay \quad (2.4)$$

Tirée de Steiner et al. (2012)

Scheduled_receive_pit représente le point de temps nominal de la permanence d'une *compressed PCF*. Hors, dans le monde réel, vu la variation des oscillateurs, les erreurs de numérisation et le passage par des domaines d'horloges différents ainsi que par des composants défaillants, il est nécessaire de spécifier un intervalle autour du *Scheduled_receive_pit* appelé "*Acceptance Window*".

Dès lors, les *compressed PCFs* qui deviennent permanentes à l'intérieur de l'*Acceptance Window* sont appelés "*in-schedule*", les autres sont des "*out-of-schedule*".

Enfin, à la fin de l'*Acceptance Window* les SMs/SCs évaluent les *in-schedule* reçues. Cela se fait selon trois étapes (SAE, 2011) : *Per-Channel Selection, Low-Membership Exclusion et Clock-Correction Calculation*.

Ces étapes visent comme leurs noms l'indiquent à sélectionner la/les *compressed PCF(s)* la/les plus pertinente(s) afin de pouvoir calculer la valeur de *Clock-Correction* qui permet d'ajuster la valeur de l'horloge locale de chaque intervenant du réseau et ainsi d'aboutir à une synchronisation globale de l'ensemble des horloges du système.

2.3.1.3 Protocole Startup/Restart

Le protocole Startup/Restart permet d'intégrer un ensemble de nœuds, prêt pour la communication, mais pas encore synchronisé à un fonctionnement synchrone déjà établi, avec un garanti de délai et ce, même en présence de nœuds ou liens défectueux.

Sa fonction est assurée grâce à un ensemble de machines d'états implémentés aux niveaux des SMs et CMs et réalise quatre fonctions: Intégration, Coldstart, Détection de cliques et Restart.

2.3.1.3.1. Integration

Conformément avec sa machine d'états (SAE, 2011) un composant mis sous tension ou réinitialisé va commencer avec l'état Intégration : initialement il va tester s'il peut intégrer une synchronisation globale existante en attendant pendant deux cycles d'intégration qui est le temps nécessaire pour recevoir, s'il en existe, une trame PCF IN. Au-delà de cette période il conclut qu'il n'existe pas une synchronisation globale disponible et exécute de ce fait la procédure Coldstart.

Les deux diagrammes suivants décrivent d'une façon simplifiée le processus d'intégration aux niveaux SMs et CMs :

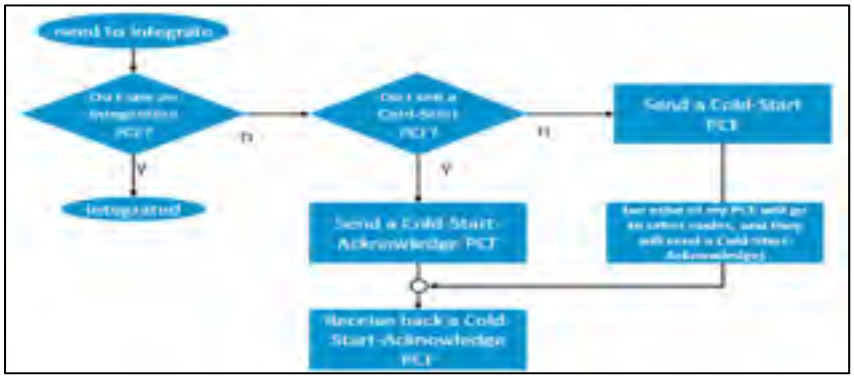

Figure 2.6 Intégration niveau Synchronization Master
Tirée de Jean-Baptiste (2013)

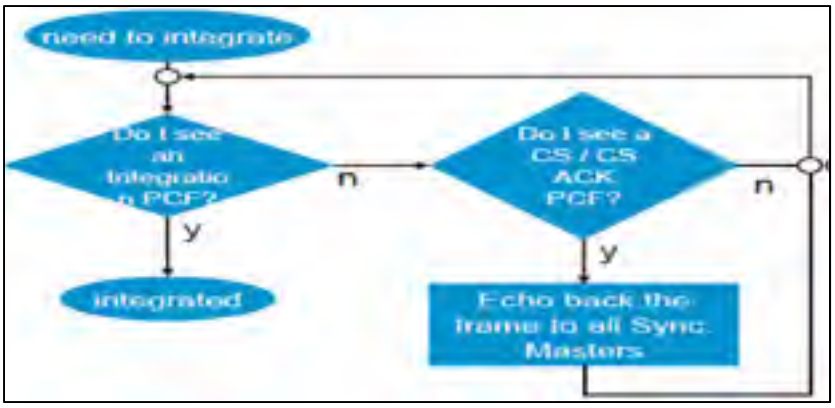

Figure 2.7 Intégration niveau Compression Master
Tirée de Jean-Baptiste (2013)

Étant aussi considéré comme un Synchronization Client, un Compression Master va s'intégrer sur la réception des PCF IN envoyés par les Synchronization Masters.

2.3.1.3.2. Coldstart

Coldstart (Jean-Baptiste, 2013; SAE, 2011) est nécessaire pour remédier au problème de transmission de messages inconsistants. Il se déroule suivant quatre étapes (voir Figure 2.8) :

- un SM envoie une trame Coldstart (CS) vers tous les CMs.
- les CMs relaient le CS vers tous les SMs du réseau. Selon des configurations de tolérance à deux fautes,
- tous les SMs accusent la réception de CS exceptant l'émetteur original, par l'envoi de la trame Coldstart Acknowledgment (CA) après un intervalle de temps configuré du point de permanence de CS reçue.
- Les CMs relaient les CA reçues vers tous les SMs du réseau.

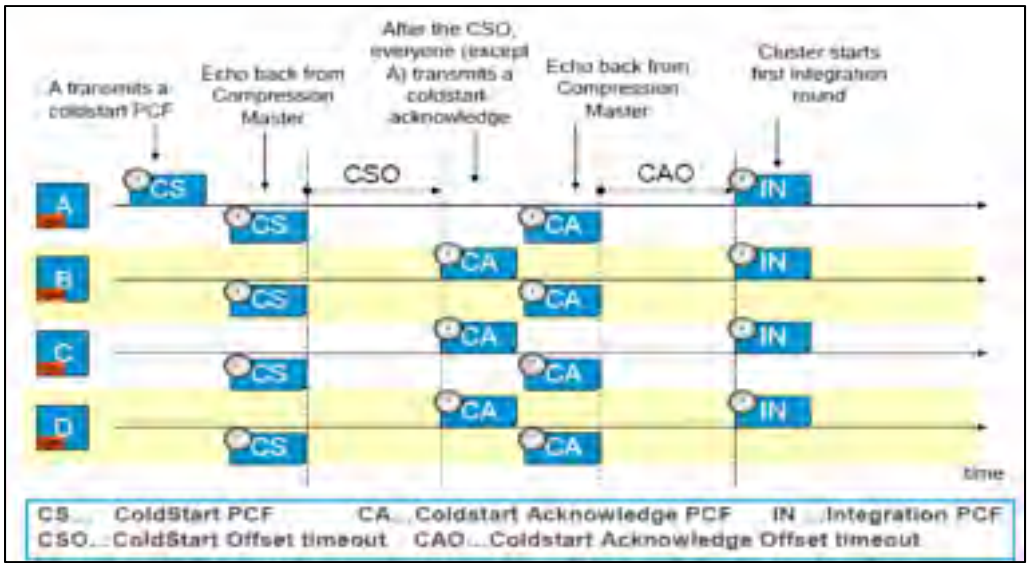

Figure 2.8 Coldstart: deux tours d'échange de messages de bout en bout
Tirée de Jean-Baptiste (2013)

Ces deux tours d'échange de messages sont indispensables dans le scénario de pire cas (deux composants défaillants : un SM et un CMs) car ils garantissent que soit la trame CS d'origine, soit la trame CA de réponse à CS va être émis par un SM correcte vers tous les CMs du réseau, condition essentielle pour pouvoir détecter et isoler un message inconsistant.

2.3.1.3.3. Restart

Dès la détection de la perte de synchronisation, le composant détecteur va essayer de regagner la synchronisation. La démarche varie selon l'état courant du composant tel qu'il est présenté par la machine d'états (SAE, 2011): elle va d'une réintégration simple si le composant est dans un état stable (SM_STABLE) à un re-Coldstart si le composant est dans un état moins stable (SM_TENTATIVE_SYNC).

2.3.1.3.4. Clique detection

La détection de cliques vise à détecter d'une façon fiable les différents scénarios de perte de synchronisation aux états synchronisés de la machine d'états (SM_TENTATIVE_SYNC, SM_SYNC et SM_STABLE).

Il existe trois algorithmes de détection de cliques, qui peuvent être exécutés en parallèle:

- **La fonction de détection de clique synchrone** : elle est essentielle pour exclure les composants défaillants hors du processus de synchronisation. Elle s'exécute à la fin de la fenêtre d'acceptance et utilise la variable local_sync_membership pour sauvegarder le champ pcf_membership_new de la trame IN reçue durant la période in-schedule et ayant le nombre maximum de bits différent de zéro. Quand ce nombre est inférieur à un seuil prédéfini, une clique synchrone est détectée.
- **La fonction de détection de clique asynchrone :** elle représente le moyen usuel pour la détection de cliques par les composants non défaillants. Elle s'exécute au début de chaque cycle d'intégration avant la transmission de la trame IN. Elle sauvegarde les champs pcf_membership_new des trames IN reçues durant la période out-of-schedule dans sa variable local_async_membership. Quand le nombre de bits différents de zéro de sa variable est au-dessus d'un seuil configuré, une clique asynchrone est détectée.
- La **fonction de détection de clique relative :** elle est essentielle pour résoudre le problème de formation de clique temporaire durant le Coldstart. Elle s'exécute au début de chaque cycle d'intégration avant la transmission de la trame IN et se manifeste quand la variable local_sync_membership est égale ou inférieure à local_async_membership.

2.3.2 Tolérance aux fautes

Théoriquement (Florin; Jean-Baptiste, 2013), pour qu'un système distribué puisse tolérer n'importe quelle faute unique (f=1) sans aucune hypothèse préalable, il lui faut au moins :

- 3f + 1 = 4 composants (3 émetteurs redondants et un récepteur) où chaque nœud peut échouer arbitrairement,
- 2f + 1 = 3 canaux de communication indépendants,
- f + 1 = 2 tours de communications (perte de la bande passante)

Il est clair de ce fait que la tolérance aux faite est un service onéreux surtout à cause le nombre élevé de composants redondants qu'il sollicite. Ce coût peut être allégé en considérant l'hypothèse "Fail-silent" pour le système en question.

Un composant est dit Fail-silent lorsqu'il soit produit un message correcte soit, en cas de défaillance, le message sera reconnu sans aucune ambiguïté par les autres composants comme étant un message fautif.

Avec cette hypothèse, les exigences pour tolérer une faute (f=1) se réduisent à :

- f + 1 = 2 composants redondants avec le garanti Fail-silent,
- f + 1 = 2 canaux de communication qui ne produisent pas de faux messages et un seul tour de communication (ce qui augmente la bande passante disponible et démunie les délais de transmission.

En important ces résultats dans le cadre du réseau TTEthernet, on aura une possibilité de tolérer jusqu'à deux fautes (dual-fault-tolerance) vu que la spécification AS6802 limite le nombre de canaux possibles à trois.

De ce fait, TTEthernet peut être configuré selon deux niveaux différents de tolérance aux fautes : le réseau peut tolérer un seul composant défaillant (soit un terminal, soit un commutateur) pour la configuration "single-failure". Deux fautes seront tolérés pour la configuration "dual-fault-tolérance" (deux commutateurs ou deux terminaux ou un commutateur et un terminal).

Quant au modèle de faute, il dépend des configurations présentées ci-dessus : pour la configuration single-failure le terminal défaillant peut avoir un comportement arbitraire (Byzantain) tandis que pour la configuration dual-fault, le comportement du terminal défaillant est plus restreint en étant un inconsistant-omission failure.

Côté commutateurs, leur modèle de faute est inconsistant-omission pour les deux configurations possibles : dans le pire cas, un commutateur défaillant peut ignorer ou échouer la réception/l'envoi d'un nombre arbitraire de messages de/vers certains de ses ports mais ne produit jamais un message invalide.

L'état inconsistant-omission-failure traduit bien l'hypothèse fail-silent discutée plus haut.
Pour passer du mode faute arbitraire à un mode inconsistant-omission failure deux moyens d'isolation de fautes sont possibles : modèles Central Guardian et High Integrity.

2.3.2.1 Le modèle Central Gardian

La fonction Central Guardian (Steiner et al., 2012) est implémentée au niveau du commutateur TTEthernet et permet au réseau TTEthernet de se protéger contre les terminaux ayant un comportement arbitraire. La figure suivante décrit le contrôle effectué par la fonction Central Guardian et les actions d'imposition (ou enforcement) appliquées aux trafics TT et RC.

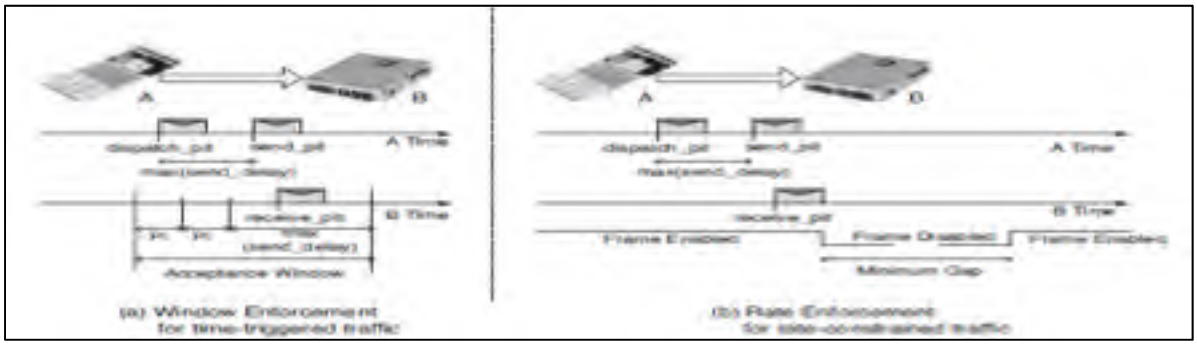

Figure 2.9 Application de Central Guardian pour des trafics TT (a) et RC (b)
Tirée de Steiner et al. (2012)

Un commutateur TTEthernet connecté à un terminal doit recevoir le premier bit de la trame TT à la point de réception receive_pit avec:

$$receive\_pit = send\_pit + link\_latency \quad (2.5)$$

Tirée de Steiner et al. (2012)

Le commutateur contrôle la fenêtre de réception (ou "acceptance window") d'une trame TT. Pour le faire, il doit considérer les conditions dans lesquelles l'*acceptance window* englobe le point de réception le plus en avance possible ainsi que celui le plus tardif. Dans ce cas le paramètre précision d'horloges (Π) doit être pris en considération. On a alors:

$$acceptance\_window\_start = dispatch\_pit + link\_latency - \Pi \quad (2.6)$$

$$acceptance\_window\_end = acceptance\_window\_start + 2 * \Pi + max(send\_delay) \quad (2.7)$$

Tirée de Steiner et al. (2012)

Quand l'algorithme d'imposition est activé pour le trafic TT, le commutateur TTEthernet ne retransmet la trame TT que si elle a été reçue durant la fenêtre de réception imposée.

Pour le trafic RC le commutateur veille à ce que le trafic RC ne dépasse pas certain débit, cela se fait en imposant une distance temporelle entre deux trames consécutive (gap) si cette distance est inférieure au gap configuré la trame ne sera pas transmise.

2.3.2.2 Le modèle High-Integrity

Ce modèle (Steiner et al., 2012) vise à confiner l'erreur localement plutôt que par l'intermédiaire d'un composant distant à l'instar du modèle Central Guardian. Il opère selon la méthode commandant/moniteur (COM/MON).

Ce modèle est illustré par la figure ci-dessous :

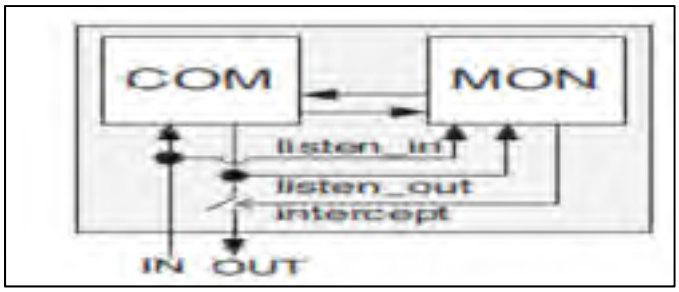

Figure 2.10 Architecture Commandant/Moniteur
pour un composant à haute intégrité
Tirée de Steiner et al. (2012)

Les deux composants se partagent un même oscillateur. Le moniteur d'horloge est dédié pour éviter une défaillance commune d'horloges. Si les deux éléments s'accordent sur l'information reçue (via l'exécution d'un protocole d'accord), et sachant qu'ils détiennent leur temporisation de la même source, l'exécution de la même machine d'état dans les composants doit produire le même résultat à environ le même instant, le cas contraire le moniteur MON intercepte la communication et termine la transmission de COM au port de sortie.

2.3.3 Discussion

La synchronisation d'horloges et la tolérance aux fautes sont des mécanismes indissociables pour les réseaux TTEthernet.

D'une part, la nature synchrone du système influe naturellement sur les procédures et dispositifs entreprises pour gérer la tolérance aux fautes. De plus, la spécification TTEthernet permet au protocole de synchronisation d'horloges de jouer un rôle tolérant aux fautes grâce au processus d'isolation des nœuds défaillants basé sur le vecteur *pcf_membership_new* de la trame de synchronisation PCF et le seuil *acceptance_threshold*.

De l'autre côté, une faute influe directement sur la qualité de synchronisation du système selon le modèle de faute en question. (Steiner et Dutertre, 2011; Steiner et Dutertre, 2010) ont mené des travaux de vérifications formelles sur cet aspect via le vérificateur de modèles

SAL. Les résultats ont montré que la synchronisation varie considérablement selon la nature des nœuds défaillants (CM ou SM) et leurs combinaisons possibles. Des scénarios peuvent être très complexes voire impossible à gérer ce qui impose de restreindre les modèles de fautes possibles et poser des hypothèses sur le fonctionnement des nœuds défaillants.

Finalement, un problème significatif existe au niveau de la simulation du protocole de synchronisation en TTEthernet compte tenu sa consommation excessive du temps de simulation. Cela revient au grand nombre d'évènements liés à la synchronisation d'horloges par rapports à ceux dédiés au fonctionnement du protocole.

(Todorov et al., 2013) a proposé une approche qu'il a qualifiée de « évolutionniste » qui consiste à enregistrer les résultats de synchronisation des simulations antérieures pour les reproduire aux modèles d'horloges des simulations qui suivent. Cette imitation du comportement de synchronisation antérieure a permis à une simulation s'exécutant sur le même réseau d'éviter le processus de synchronisation en temps réel ce qui a permis de gagner jusqu'à 40 fois de temps de simulation selon (Todorov et al., 2013).

2.4 Les différents modèles de simulation de TTEthernet

Actuellement, il existe deux implémentations de modèles de simulation de TTEthernet : une extension INET d'OMNeT++ (Steinbach et al., 2011) et une implémentation OPNET (Abuteir et Obermaisser, 2013).

2.4.1 Modèle de CoRE4INET

Ce modèle se base sur la plateforme OMNeT++ ainsi que INET framework qui implémente les principales fonctionnalités de réseaux. Sa réalisation passe par plusieurs modifications et extensions apportées au modèle Ethernet de base d'INET framework.

La figure suivante représente l'intégration des composants majeurs du modèle conçu à l'intérieur d'une implémentation INET standard.

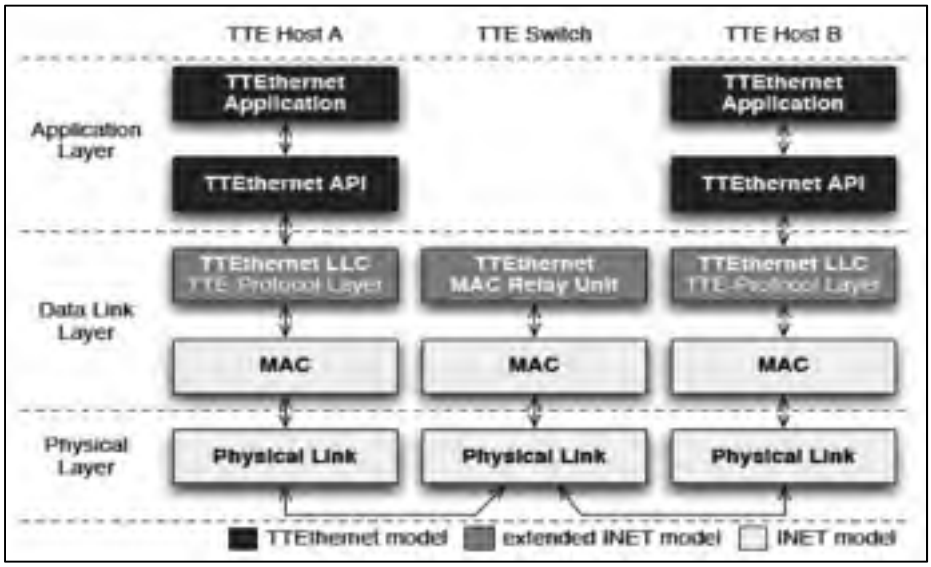

Figure 2.11 Intégration de TTEthernet dans INET
Tirée de Steinbach et al. (2011)

2.4.1.1 Concepts et modèles

La modélisation d'un réseau TTEthernet pour sa version extension INET mène à la conception de trois modèles de base : le modèle horloge, le modèle commutateur et le modèle terminal.

2.4.1.1.1. Modèle d'horloge

L'horloge est un élément fondamental pour le processus de synchronisation TTEthernet. Elle se base dans son fonctionnement sur des "*ticks*" Le temps entre deux *ticks* est configurable. Toutes les horloges ont une certaine imprécision appelée "*clock drift*" ou déviation d'horloge. Ces imprécisions peuvent influencer amplement le comportement du protocole et doivent être prise en charge par le modèle à implémenter.

Dans ce modèle, plusieurs *ticks* sont simulés en même temps car il est impossible de modeler la déviation "*drift*" en simulant chaque *tick* à part, vu le grand ralentissement que va subir la simulation.

Un facteur de déviation "*drift factor*" est spécifié afin de prendre en charge la déviation de l'horloge. Il est supposé constant pour un intervalle de temps configurable.

La formule de calcul de l'horloge est, par conséquence, la suivante :

$$t' = t + \delta \ * (\Delta t_{Tick} + \Delta t_{Drift}) \tag{2.8}$$

Tirée de Steinbach et al. (2011)

où t' est le temps de l'évènement suivant, t est le temps actuel de simulation, δ est le nombre planifié de *ticks*, pour un évènement donné, Δt_{Tick} est la durée par un seul *tick* et Δt_{Drift} est la moyenne de déviations "*average drift*".

Pour conclure, ce modèle constitue une simplification valide et précise du fonctionnement de l'horloge, puisque dans le monde réel la variation de la moyenne de déviation est très basse pour un seul cycle.

La synchronisation de l'horloge est renforcée d'avantage à l'aide de messages cycliques PCFs qui permettent, le cas échéant, de resynchroniser l'horloge à l'intérieur d'un cycle.

2.4.1.1.2. Modèle du commutateur TTEthernet

Un commutateur TTEthernet doit avoir les moyens de gérer le trafic TT, RC et BE. Cela se fait à travers l'extension du commutateur Ethernet standard par un autre module ayant les outils nécessaires pour manipuler des trafics critiques. On en cite, un ordonnanceur pour gérer le plan de transmission, une horloge locale, un protocole de synchronisation et un autre protocole de transmission de trafics critiques.

Le modèle de commutateur TTEthernet à implémenter contient aussi une unité nommée MACRelayUnit standard d'INET pour gérer le trafic BE.

Le module traitant les trafics critiques incorpore un protocole de synchronisation faisant référence à l'horloge locale décrite précédemment. Sa logique de transmission consiste à classifier les paquets selon leurs types en se basant sur l'adresse de destination de chacun d'entre eux.
Une décision de transmission sera prise selon ce type :

- Les messages PCFs seront évalués par le module de synchronisation du commutateur.
- Les messages TT seront stockés dans une file d'attente jusqu'à leurs temps d'envoi planifiés.
- Les messages RC seront transmis le plus tôt possible, mais avec une priorité plus faible que celle des messages TT.
- Les messages BE seront transmis pendant le temps qui reste.

D'un autre côté, à la réception, le commutateur TTEthernet contrôle la conformité des trames avec certaines contraintes. Celles-ci se situent dans une table préconfigurée du commutateur appelée *CTC-Table.* Ce contrôle revêt une importance cruciale en protégeant le système d'être corrompu par des émetteurs défectueux qui ne respectent pas les contraintes temporelles.

Plus précisément, la tâche de contrôle consiste à vérifier l'arrivée d'un message au bon moment et au bon port.
Le contrôle au niveau d'un message RC se fait à travers la vérification de la concordance avec les comptes d'écart d'allocation de la bande passante (*gap*), alors que pour un message TT, le commutateur vérifie s'il arrive au temps planifié.

La vérification des ports de réception, permet de détecter les messages erronés provenant des émetteurs défectueux, même s'ils arrivent à se comporter comme des émetteurs TT. Ces messages sont, par conséquence, rejetés.

Concernant les trames BE, elles sont tolérées à n'importe quel port et à n'importe quel moment. Cela permet, ainsi, l'ajout au système d'autres composants qui ne tiennent pas copmte du protocole TT.

2.4.1.1.3. Modèle du terminal TTEthernet

L'implémentation d'un terminal TTEthernet nécessite également l'extension du protocole Ethernet d'INET avec les services TTEthernet. Il faut donc incorporer un ordonnanceur, un module de synchronisation, ainsi qu'un module de classification de messages.

La mission de l'ordonnanceur consiste à déclencher l'envoi et la réception de messages en concordance avec un plan de transmission de messages.

Quant au module de classification, comme son nom l'indique, il permet de classifier les messages reçus selon leurs types et les traiter, toujours, en concordance avec le plan de transmission de messages.

2.4.1.2 Implémentation

L'un des points forts de TTEthernet est son adaptabilité avec l'existant. Le modèle implémenté préserve cet aspect. Il respecte, de plus près, les concepts d'INET standard : l'héritage est utilisé quand un ajustement des modules INET est possible. Le cas échéant, si l'existant ne peut pas satisfaire les besoins spécifiques de TTEthernet, des nouveaux modules seront implémentés.

2.4.1.2.1. Commutateur TTEthernet

D'un point de vue fonctionnel, un commutateur TTEthernet peut être divisé en deux unités logiques de transmission. La première gère le trafic BE alors que la deuxième s'occupe de la gestion des trafics critiques (TT et RC).

De plus, ces deux unités se partagent les mêmes ports physiques. Elles veillent sur le respect des priorités des trafics ainsi que de la politique d'accès au support. Cela se fait grâce à des modules spéciaux qui les séparent de la couche MAC.

Le modèle implémenté respecte cette composition : il hérite de l'INET standard son fondement autour d'une unité de base appelée *MACRelayUnitTTEthernet* (voir figure 2.12). Celle-ci est, à son tour, divisée en deux unités *MACRelayUnit.* La première gère le trafic BE alors que la deuxième se focalise sur la gestion des trafics critiques (TT et RC).

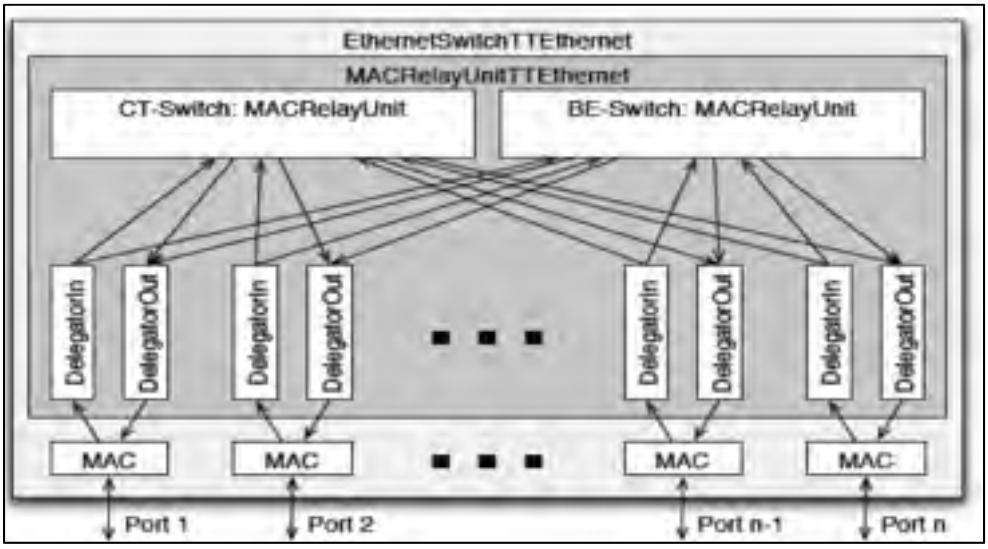

Figure 2.12 Modèle d'un commutateur avec deux unités de transmission (MACRelayUnits) et deux Delegators par port

Tirée de Steinbach et al. (2011)

L'accès à la couche MAC pour ces unités se fait à travers deux modules appelés *Delegators*. Le *DelegatorIn* est responsable à l'acheminement des trames provenant du module *EtherMAC* vers l'unité *MACRelayUnit* adéquate du commutateur (celle qui assume le fonctionnement de CT-Switch ou BE-Switch).

La décision de l'acheminement est prise selon le type du trafic. Celui-ci est déterminé par le champ adresse de destination, conformément avec les spécifications du protocole TTEthernet (SAE, 2011).

D'un autre côté, pour les trames sortantes, le *DelegatorOut* reçoit les trames à partir des deux unités de transmission et envoie-les vers la couche MAC.

Aux niveaux des unités de transmission, un principe fondamental de TTEthernet est appliqué : c'est le fait de ne jamais retarder un trafic TT par d'autres types de trafics. Ce mécanisme incombe au *DelegatorOut* qui s'assure de l'acheminement des trames selon les priorités adéquates, c.à.d. il veille à chaque fois que le temps restant avant la prochaine émission d'une trame TT (mode réservé) est suffisant pour pouvoir transmettre entièrement un message RC ou BE. Le cas échéant, ces deux types de trafics seront stockés dans des files d'attentes indépendantes.

A ce niveau, il est important de mentionner que le problème de transmission des trames TT se pose au niveau de file d'attente du module *EtherMAC* : même si le *DelegatorOut* est en mode réservé, il est possible qu'un message TT soit retardé par d'autres trames de la file d'attente de la couche MAC.

La solution envisagée par le modèle consiste à empêcher le *DelegatorOut* d'envoyer des trames à la couche MAC tant que sa file d'attente est utilisée. Ce comportement nécessite évidemment que l'*EtherMAC* communique l'état de sa file d'attente avec le *DelegatorOut.*

Cela se fait à travers le déclenchement, par l'*EtherMAC*, d'un évènement informant son correspondant (le DelegatorOut) qu'il a fini l'envoi d'un message et que son interface est de nouveau disponible.

Le problème avec cette solution est que l'implémentation standard de l'*EtherMAC* ne permet pas l'échange de tels messages (déclencheurs d'évènements) entre les différents modules. Dans ce cas, on doit passer par la table de notification d'INET (*INET Notification Board*). Cette table représente un système éditeur/abonné (*publisher/subscriber system*), où chaque module est autorisé à placer des notifications ou encore, enregistrer des évènements qui seront livrés par la suite suivant une méthode de rappel (*callback method*). Les modules *EtherMAC* peuvent y mettre plusieurs évènements contenant l'état actuel de chacun d'entre eux. Exemple, le message NF_PP_TX_END indique la fin de livraison d'une trame. Sur réception d'un tel évènement, le *DelegatorOut* vérifie si la file d'attente de la couche MAC est libre. Si c'est le cas, il procède à l'envoi de sa prochaine trame.

La solution proposée nous permet de conserver les modules standards d'*EtherMAC* en comblant leur insuffisance par un mécanisme d'échange de messages.

A la fin, une idée intéressante pouvant offrir une amélioration notable à la faveur du service Temps-réel du commutateur a été évoquée. Elle consiste d'implémenter une version adaptée du module EtherMAC ne contenant pas de file d'attente et qui envoie directement un message de disponibilité au DelegatorOut.

La figure 2.13 montre une implémentation OMNeT++ d'un commutateur TTEthernet. Elle comporte trois parties : la première présente une vue globale de la structure d'un commutateur TTEthernet (les modules EtherMACs, la table de notification et l'unité de transmission TTE*MACRelayUnit*), la deuxième partie spécifie le contenu de TTE*MACRelayUnit* (les Delegators In/Out et les unités de transmission pour les de trafiques CT/BE). Encore plus spécifique, la troisième partie précise le contenu de l'unité de

transmission pour le trafic critique *MACRelayUnitCT*, on y trouve aussi une logique de transmission et une horloge qui coordonne cette logique.

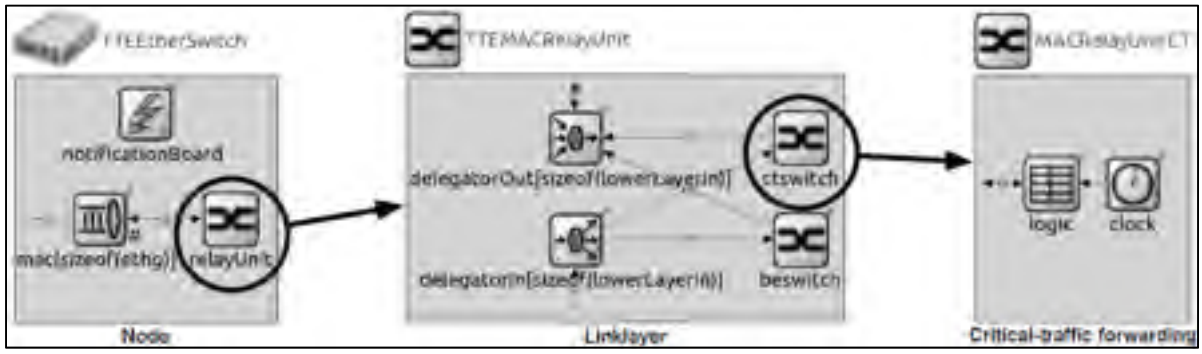

Figure 2.13 Un aperçu de la structure d'un modèle du commutateur TTEthernet
Tirée de Steinbach et al. (2011)

2.4.1.2.2. Terminal TTEthernet

L'implémentation de la pile de protocolaire d'un terminal TTEthernet nécessite l'ajout de deux extensions majeures à ceux d'Ethernet standard : une extension de la sous-couche LLC d'INET et l'implémentation de l'interface *TTEthernetAPI* pour les besoins de simulation.

2.4.1.2.2.1. Sous-couche LLC de TTEthernet

Pour notre modèle de commutateur TTEthernet, la sous-couche LLC a été enrichie par rapport à INET standard en lui implémentant, en plus, les services de TTEthernet (voir figure 2.14). Son travail se fait en parfait collaboration avec l'ordonnanceur : elle se connecte à lui et répond à ses commandes.

De plus, cette sous-couche réserve deux types de files d'attente pour les messages TT. L'une se sert lors de l'envoi, l'autre à la réception des trames TT. Une file d'attente est identifiée par l'identificateur CT-ID du message.

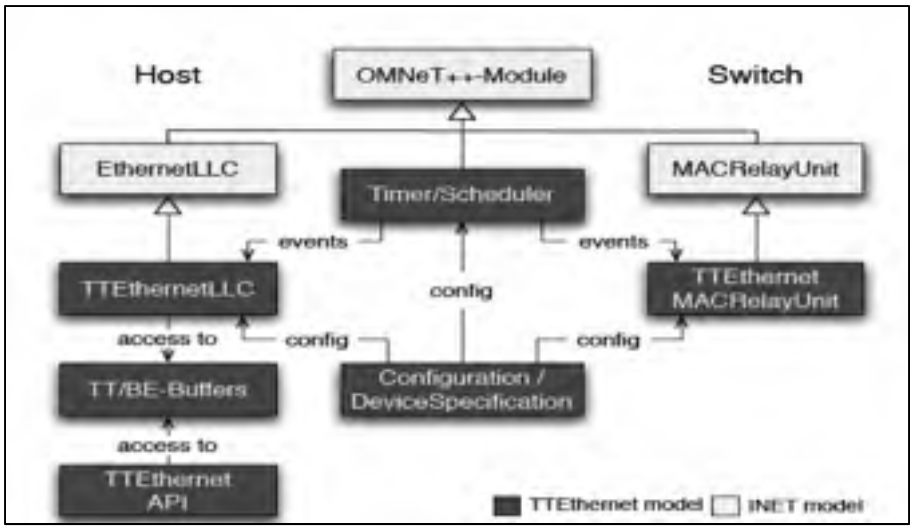

Figure 2.14 Modèle objet des couches du protocole TTEthernet
Tirée de Steinbach et al. (2011)

L'activité d'un ordonnanceur est basée sur l'échange de messages. Il fonctionne comme suit : si le temps d'émission d'un message TT arrive, il envoie la commande *TTOutgoing traffic* avec l'identificateur CT-ID du message, à la sous-couche LLC. En réponse à cette commande, la sous-couche LLC, en s'appuyant sur l'identificateur CT-ID, lit le message à partir de la file d'attente, le formate et le transmet vers la sous-couche MAC.

Enfin, il est important de noter que les applications TT se connectent avec un plan de transmission (*schedule*) via un évènement spécial envoyé par l'ordonnanceur. Cela permet à ces applications de fournir les données immédiatement avant la transmission.

2.4.1.2.2.2. TTEthernet API

TTEternet API est définie par TTTech. Elle gère l'envoi et la réception des messages au niveau application de TTEthernet. Comme présenté par la figure 2.11, *TTEthernet API* connecte la couche application de TTEthernet à la sous-couche LLC.

L'échange de données avec la sous-couche LLC est réalisé à travers une méthode d'accès aux files d'attente basée message (*Buffer_based_message access*). Cette méthode permet un accès par octet (*byte wise*) à la charge utile du message et permet une optimisation des performances temps-réel.

D'un autre côté, chaque application a un ou plusieurs gestionnaire(s) (*handler*) contrôlant l'accès aux files d'attente. Chaque gestionnaire est souvent déterminé à la phase d'initialisation de l'application et inclus un contrôleur d'identité "*Controller ID*" pour identifier les files d'attente, le type de trafic pour classifier les messages et une indication du sens de la file (envoi ou réception).

Par conséquence, afin d'envoyer ou recevoir un message, une application doit spécifier les trames (information de l'entête et l'identité de la file d'attente pour stocker la charge utile), ainsi qu'un gestionnaire de la file d'attente.

A la fin, il faut noter que l'implémentation de *TTEthernet API* d'INET permet d'intégrer des applications réelles durant le processus de développement et de test ce qui permet de renforcer l'efficacité et la validité externe des systèmes implémentés.

2.4.2 Modèle TTEthernet pour OPNET

Le modèle TTEthernet pour l'environnement de simulation OPNET (Abuteir et Obermaisser, 2013) est organisé hiérarchiquement en trois niveaux de base : le niveau réseau, le niveau nœud et le niveau processus.

Le niveau réseau permet de de créer le modèle du réseau à l'aide soit de blocs prédéfinis de sa bibliothèque standard, soit des composants définis par l'utilisateur. C'est à ce niveau que se déroule la collecte des statistiques et l'affichage de résultats.

Le deuxième niveau définit le modèle des nœuds à partir d'une structure modulaire. Chaque nœud se construit à partir de la connexion de plusieurs modules. Cette connexion permet l'échange des messages et d'autres informations d'état. Ce modèle définit deux composants du réseau TTEthernet qui sont les commutateurs et les terminaux.

Le dernier niveau de la hiérarchie s'occupe de l'implémentation des modules en utilisant des modèles de processus représentés par des machines à états finis, des définitions des fonctions du modèle et une interface définissant les paramètres à interfacer avec les autres modèles de processus ainsi que les attributs de configuration. Ces modules définissent les couches protocolaires à l'intérieur des commutateurs et des terminaux TTEthernet.

2.4.2.1 Modèle du terminal TTEthernet

Le fonctionnement d'un terminal TTEthernet est assuré grâce à la coordination de ses différents blocs fonctionnels présentés ci-dessous :

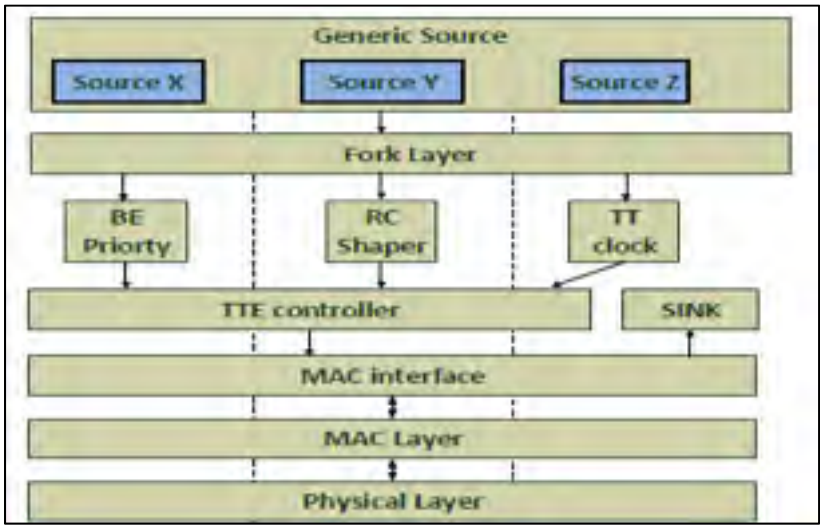

Figure 2.15 Diagramme des blocs d'un terminal TTEthernet
Tirée de Abuteir et Obermaisser (2013)

La source générique génère la charge utile des différents trafics avec un timing qui correspond aux spécificités de chaque application (bande passante, latence). Des outils de planification sont utilisés pour calculer un plan de communication pour les trames TT et les paramètres nécessaires pour les trames RC tels que la gigue et BAG. Le plan de communication est enregistré aux niveaux des terminaux et des commutateurs et permet de configurer ses modèles.

La source générique passe la charge utile à la couche "*Fork*" qui à son tour passe la trame reçue vers la couche "*TT clock*" où elle sera reconnue et stockée dans le buffer correspondant à son VL. Elle sera envoyée ensuite vers la couche "*TTE Controller*" au temps spécifié par le plan de communication statique.

La couche *TTE Controller* contient trois files d'attente, une pour chaque type de trafic. La trame venant de la couche *TT clock* sera envoyée immédiatement vers la couche inférieure au cas où aucune trame n'est en cours de transmission, le cas échéant, la trame sera conservée dans la file d'attente correspondante. L'envoi des trames vers l'interface MAC se fait selon l'ordre de priorité attribué à chaque type de trafic.

Pour une trame RC, après être reçue et reconnue par la couche *Fork* à l'aide du port de réception, elle sera aiguillée vers la couche *RC shaper* responsable à la régulation du trafic RC : un intervalle minimal BAG est garanti entre deux instances de trames RC consécutives pour chaque lien virtuel. Ces instances de trames sont stockées par conséquent dans des files d'attente dédiées à chaque lien virtuel avant d'être acheminées vers l'un de deux files d'attente multiplexées et contrôlées par le BAG selon leur ordre de priorité. Dès leurs réceptions par la couche *TTE controller*, ils sont mis dans la file d'attente dédiée au trafic RC.

Pareille pour le trafic BE, après sa réception de la part de la couche *Fork*, les trames sont transmises vers la couche *BE Priority* qui les affecte à l'une de deux files d'attente de priorité. En respectant la priorité de chaque trame, elles sont transmises, après, vers la file BE

de la couche *TTE controller* qui s'occupera, comme pour tous les autres types de trafics, de les diriger vers l'*interface MAC.*

Comme son nom l'indique, l'interface MAC joue le rôle d'interface entre les trames sortant de la couche supérieur et ceux entrant de la couche MAC. Les trames venant de la couche MAC seront accueillies par la couche SINK qui les traite et s'occupe de la collection des statistiques.

2.4.2.2 Modèle du commutateur TTEthernet

Un commutateur TTEthernet est essentiellement l'extension d'un commutateur Ethernet standard avec une fonction de transmission déterministe pour les trames TT et éventuellement, une fonction pour la gestion des trames RC. Ces fonctions sont implémentées à l'aide d'une couche additionnelle appelée *Pont* située au-dessus de la couche MAC. La nouvelle structure du commutateur est explicitée par la figure 2.16.

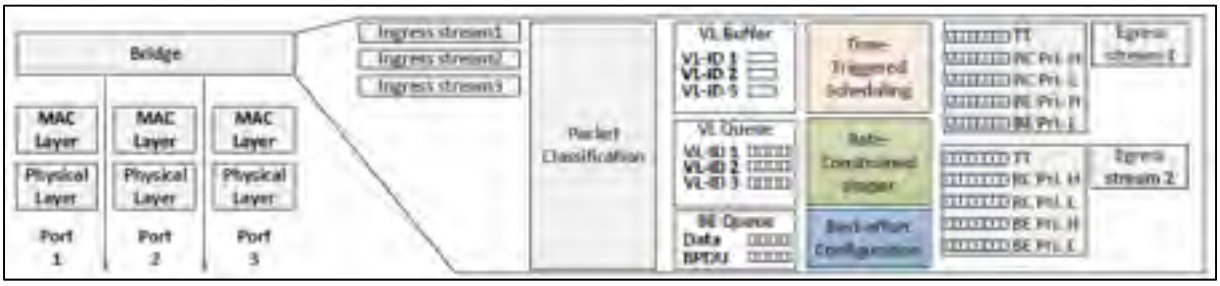

Figure 2.16 Diagramme de blocs d'un commutateur TTEthernet
Tirée de Abuteir et Obermaisser (2013)

Un commutateur TTEthernet est doté de plusieurs ports associés chacun à une couche physique et une couche *MAC* enrichie. La couche MAC vérifie la validité de l'adresse de destination des trames entrant en extrayant le champ constant de l'adresse de destination et en le multipliant avec un masque adéquat. Si le résultat est équivalent à une certaine valeur *CT marker* prédéfinie alors il s'agit d'un trafic critique (TT ou RC), la différentiation entre eux se fait alors selon la valeur du champ *EtherType*. Sinon trame est BE.

Le pont permet de manipuler et de transmettre les trames entrantes vers les ports de sortie selon le type de trafic. Ce fonctionnement se base sur cinq couches :

- la couche de classification (*Classification layer*) : détermine le type de la trame entrante (TT, RC ou BE)
- la couche de planification TT (*TT scheduling layer*)
- la couche de vérification des trames RC (*RC shaper layer*)
- la couche de configuration des trames BE (*BE configuration layer*)
- la couche des ports de sortie (*Egress port layer*) : s'occupe de la politique d'intégration de flux (shuffling/timely bloc)

Pour chaque trame TT un plan de communication définit les trois paramètres de période, de phase et de longueur, en plus d'autres informations concernant le port d'entrée, le port de sortie et la taille du buffer du *pont*.

Initialement, la couche de classification vérifie la validité et l'intégrité des trames TT arrivant à la couche MAC. Cela se fait à travers la vérification que la taille de la trame et le port d'entrée sont conformes à ce qui est décrit dans le plan de communication du lien virtuel (*VL*) de la trame. Une fois validée, elle sera stockée au *buffer* correspondant à son VL. Ce dernier fournit un espace mémoire (*buffer*) pour exactement une trame TT. Si le *buffer* est occupé et une autre trame TT arrive, la nouvelle trame remplace l'ancienne.

En concordance avec le paramètre phase du plan de communication, la couche de planification assure la transmission de la trame TT du *VL buffer* vers la file des trames TT au niveau du port de sortie. A chacun de ces ports est associé cinq files d'attente de priorités décroissantes (une pour les trames TT, deux pour les trames RC et deux pour les trames BE).
Le rôle de la couche des ports de sortie est de transmettre les trames des files de sortie vers la couche MAC en respectant les priorités et le mécanisme d'intégration de trafics appliqué (shuffling ou timely block).

Concernant le trafic RC, ses trames sont associées à deux paramètres : le GAP qui est l'intervalle de temps minimal entre l'émission de deux trames RC consécutives sur le même VL et la gigue engendré essentiellement par le multiplexage des VL dans une file de sortie partagée. D'autres informations sont également associées à une trame RC telles que le port d'entrée, le port de sortie et la taille requise en file.

Une trame RC sera vérifiée à son arrivée au pont par l'unité de filtrage de la couche de classification. Sa taille doit être inférieure à la taille maximale permise et son port d'entrée doit correspondre avec les paramètres de configuration de son VL. Les trames valides seront stockées dans la file associée à leur VL.

La couche *RC shaper* contrôle le trafic des trames RC en vérifiant le BAG des trames consécutives d'un VL et en les déplaçant, ensuite, de la file d'attente de chaque VL vers l'un des deux files de sortie selon la priorité de chacune.
La couche des ports de sortie est responsable à transmettre les trames RC des files d'attentes des ports de sortie vers la couche MAC.

Enfin, le commutateur TTEthernet traite deux niveaux de priorités pour les trames BE : le flux *BPDU (Bridge Protocol Data Units)* du protocole *STP (Spanning Tree Protocol)* transporte des informations de signalisation permettant d'établir une topologie sans boucle(s) pour les trames BE. Le deuxième flux BE géré est celui de trames de données. Deux files d'attentes sont alors requise par la couche de classification, une pour chaque type de trames.

La couche de configuration opère selon deux processus. Le premier gère les trames de données BE alors que le deuxième manipule les trames BPDU selon le protocole STP. Chacun des deux processus relaie les trames BE de l'une des files d'attente BE vers celles du port de sortie selon leur priorité. Elles seront transmises à la fin, par la couche des ports de sortie, vers la couche MAC.

2.4.2.3 Blocs d'injection de fautes

Le modèle TTEthernet d'OPNET dispose de blocs d'injection de fautes qui peuvent être instanciés et configurés afin de pouvoir injecter un message fautif particulier. Cela permet d'évaluer la fiabilité du système et tester les mécanismes de tolérances aux fautes. Les blocs implémentés traitent types de fautes suivantes :

- ***Omission failure*** **:** l'émetteur n'est pas capable d'émettre une trame ou le récepteur n'est pas capable d'en recevoir. Le module gérant ce type de fautes utilise l'outil *packet discarder* d'OPNET pour l'implémenter.
- ***Corruption*** **:** implique le changement des données d'origine. Cette faute est implémentée en utilisant l'outil *link configuration* d'OPNET qui permet d'appliquer un taux d'erreurs sur un lien spécifique et pour une durée précise.
- ***Link failure*** **:** provoque une défaillance transitoire d'un lien de transmission. Il est faisable à l'aide de l'outil *failure/recovery* d'OPNET.
- ***Crash failure*** **:** se traduit par une défaillance transitoire ou permanente d'un ou plusieurs terminaux. Il est applicable via l'outil failure/recovery d'OPNET.
- ***Delay failure*** **:** les commutateurs ou les terminaux défaillants peuvent retarder la transmission de trames. Cette défaillance apparaît après la couche *shaper* chez les terminaux et nécessite l'implémentation d'un module spécifique (*failure module*) intégré à la couche MAC pour les commutateurs.
- ***Stuck at failure*** **:** engendre l'envoi d'une trame de façon répétitive. Elle est implémentée dans le module *failure module* intégré à la couche MAC.
- ***Babbling Idiot failure*** **:** se manifeste par un terminal ou un commutateur qui envoie des trames à des instants inopportuns. Cette faute est réalisée avec la diffusion d'une forte charge des trames BE ou la génération arbitraire des trames TT ou RC sur un lien virtuel donné.
- ***Masquerading failure*** **:** se produit lorsqu'un terminal défaillant usurpe l'identité d'un autre terminal. Cela se fait pour les trames TT et RC par l'envoi d'une identité du lien

virtuel (*VL ID*) incorrecte et l'envoi des trames avec l'adresse MAC erronée pour le trafic BE.

2.4.3 Discussion

Indépendamment de l'environnement de simulation utilisé, les deux implémentations du modèle TTEthernet conservent la structure de base assurant un fonctionnement tel qu'il est décrit dans la spécification AS6802 : un modèle de commutateur et de terminal avec des contrôleurs de trafics pour vérifier et recevoir/acheminer les trames de différentes classes de trafics. Des entités mémoire (buffers et files d'attente) pour stocker séparément les trames selon un système de priorités attribuées à chaque type de flux et un ordonnanceur pour coordonner le travail des différents éléments suivant un plan de transmission préétabli.

Cependant, quelques différences de conception peuvent être décelées pour les deux modèles. En effet, contrairement au modèle basé INET, le modèle TTEthernet d'OPNET est doté d'un bloc d'injection de fautes qui lui permet de gérer avec plus de précision et efficacité des scénarios de simulations ciblant un type spécifique de fautes.

L'aspect critique de ce dernier modèle réside dans le fait qu'il ne définit pas un modèle d'horloge, pourtant crucial pour la simulation d'un modèle TTEthernet vu que c'est à partir de lui qu'on spécifie la qualité de synchronisation et par conséquent les performances atteignables par le système.

Un autre point discutable à propos du travail (Abuteir et Obermaisser, 2013), c'est que les auteurs se contentent de présenter des schémas fonctionnels du modèle de commutateur et terminal sans aborder aucun problématique niveau implémentation ni s'appuyer sur des captures écrans des blocs construits ! Ce qui met en question le degré d'avancement et l'utilisabilité du modèle vu qu'à ce jour il n'y a pas de modèles TTEthernet rendus publics pour l'environnement OPNET.

Enfin, il est important de signaler que, pour sa part, le modèle TTEthernet de l'extension d'INET présente un inconvénient incontournable qui est l'absence d'un outil de planification de tâches. C'est à la charge de l'utilisateur de le calculer et de l'entrer ensuite via des paramètres de configuration, ce qui n'est pas évident pour nombreux scénarios.

2.5 Programmation par contraintes et planification globale de tâches dans un réseau TTEthernet

L'élaboration d'un plan de transmission dépend de plusieurs contraintes régissant un réseau TTEthernet.

Un problème de planification peut être spécifié formellement sous forme de systèmes de contraintes. Cette spécification peut être traduite aisément en un langage logique avec contraintes à l'aide des outils de programmation par contraintes (exemple Yices) dotés de mécanismes puissants d'interprétation de contraintes arithmétiques et symboliques permettant ainsi la vérification de la satisfiabilité du problème et l'élaboration de modèle(s) de solutions.

2.5.1 Programmation par contraintes

La programmation par contraintes (PPC) est une technique de résolution de problèmes inspirée de l'intelligence artificielle qui est apparue dans les années 1980. Elle a pour vocation de résoudre n'importe quel type de problèmes combinatoires et est déjà utilisée pour un très grand nombre d'applications réelles telles que la configuration et diagnostic d'équipements, l'ordonnancement de lignes de production, l'affectation de fréquence et de la bande passante ainsi que pour le dimensionnement et modélisation des réseaux.
PPC s'articule autour de quatre principes : le réseau de contraintes (P), les algorithmes de filtrage, un mécanisme de propagation et un autre de recherche de solutions.

2.5.1.1 Le réseau de contraintes

Un réseau de contraintes P est composé de :

- Un ensemble fini de variables $V= \{V_1, \ldots, V_n\}$
- Un ensemble fini de domaines $D= \{D(V_1), \ldots, D(V_n)\}$ où $D(V_i)$ est l'ensemble fini des valeurs possibles de V_i.
- Un ensemble fini de contraintes $C= \{C_1, \ldots, C_m\}$ où chaque contrainte Ci est défini par :
  - un sous-ensemble de variables $var(C_i) = \{Vi_1,..., Vi_{ni}\}$;
  - une relation $rel(Ci) = rel(Vi_1,..., Vi_{ni}) \subseteq Di_1 \times Di_2 \times ... \times Di_{ni}$.

Une contrainte est une relation logique entre différentes variables, chacune prenant ses valeurs dans un domaine. Ainsi, une contrainte restreint les valeurs que peuvent prendre simultanément les variables.

Elle peut être définie en extension avec l'énumération des tuples de valeurs appartenant à la relation ou en intension en utilisant des propriétés mathématiques connues. Son arité est le nombre de variables sur lesquelles elle porte. On différencie alors des contraintes unaires, binaires jusqu'au n-aire, le cas où elle met en relation un ensemble de n variables. Elle est dite dans ce dernier cas contrainte globale. Par exemple, distinct(E) est une contrainte globale en Yices qui contraint toutes les variables d'un ensemble E à prendre des valeurs différentes.

Enfin, il est indispensable de spécifier les types de contraintes vu que le choix de la théorie du solveur va en dépendre par la suite. Le type d'une contrainte se définit en fonction des domaines de valeurs de ses variables. On distingue essentiellement :

- des *contraintes numériques*, eux même se divisent en *contraintes numériques sur les réelles* si ses variables sont des réelles et *contraintes numériques sur les entiers* au cas où ses variables ne peuvent prendre que des valeurs entières. On peut aussi les classer en *contraintes numériques linéaires* quand les expressions arithmétiques exprimant leurs

relations sont linéaires ou encore en *contraintes numériques non linéaires* quand les expressions arithmétiques contiennent des produits de variables, des fonctions logarithmiques, des exponentielles, …

- des contraintes booléennes, ils sont très répandus (en domaine de circuits logiques par exemple) et portent sur des variables booléennes. Une contrainte booléenne peut alors être exprimée sous forme d'une implication (=>), d'une équivalence (<=>) ou d'un non équivalence (<≠>) entre deux expressions.

2.5.1.2 Les algorithmes de filtrage

Appelé également algorithme de réduction de domaines. La PPC utilise pour chaque contrainte une méthode de résolution spécifique à elle, afin de supprimer pour chaque variable les valeurs des domaines impliqués dans la contrainte et qui ne peuvent appartenir à aucune solution de cette contrainte, compte tenu des valeurs des autres domaines. Ce mécanisme est assuré grâce à un algorithme de filtrage. En lui appliquant sur chaque contrainte d'un modèle donné, les domaines de ses variables vont se réduire.

Une propriété intéressante d'un algorithme de filtrage est la *consistance d'arc*. Un algorithme de filtrage associé à une contrainte est *arc-consistant* s'il supprime toutes les valeurs des variables impliquées dans la contrainte qui ne sont pas consistantes avec celle-ci. Par exemple, pour la contrainte x+5 = y avec les domaines D(x) = {1, 2, 4, 5} et D(y) = {2, 4, 6, 9, 11}, un algorithme de filtrage arc-consistant modifiera les domaines pour obtenir D(x) = {1, 5} et D(y) = {6, 9}.

2.5.1.3 Mécanisme de propagation

Après la modification du domaine d'une variable par l'algorithme de filtrage, il est utile de réétudier les conséquences de cette modification sur les autres contraintes impliquant cette variable dans le sens où la réduction du domaine d'une variable peut permettre de déduire que certaines valeurs d'autres variables n'appartiennent pas à une solution. Cela peut

conduire éventuellement à de nouvelles déductions. On dit dans ce cas qu'une modification a été propagée.

Ce mécanisme de propagation se répète à la suite de chaque algorithme de filtrage et vise à obtenir des déductions globales de domaines.

2.5.1.4 Modélisation et mécanisme de recherche de solutions

En PPC, plusieurs outils sont mis en œuvre pour solutionner un problème de satisfiabilité. Des solveurs comme Yices et MathSat utilisent une conjonction d'algorithmes et de techniques tels que présentés précédemment (algorithme de filtrage, mécanisme de propagation, backtracking) pour accomplir cette tâche. Mais, cela dépend d'abord d'une modélisation du problème.

Dans cette optique, on définit ce qu'on appelle communément un problème de satisfaction de contraintes (CSP). Un CSP est un problème modélisé sous forme d'un ensemble de contraintes posées sur des variables : il est défini en fonction du réseau de contraintes P et du triplet (V, D, C) avec V est l'ensemble de variables, D est l'ensemble de domaines finis et C l'ensemble de contraintes.

La modélisation du problème sous forme CSP cherche à lui spécifier formellement : il s'agit tout d'abord d'identifier les inconnues du problème (les variables V), ainsi que D qui définit l'ensemble de valeurs que prendra chaque variable de V. Il faut ensuite identifier les contraintes C entre les variables et ce, selon trois types possibles :

- contraintes prédéfinies du solveur (contraintes arithmétiques, de cardinalité, …)
- contraintes en extension
- Meta-contraintes qui sont des combinaisons de plusieurs contraintes reliées par des opérateurs logiques ET, OU, XOR, NOT.

La résolution d'un CSP consiste à affecter des valeurs aux variables de telle sorte que les contraintes soient respectées. Plus précisément, une *affectation* est le fait d'instancier certaines variables par des valeurs, elle peut être totale, si elle instancie toutes les variables du problème ou partielle si elle n'instancie qu'une partie. Elle est *consistante* si elle ne viole aucune contrainte, *inconsistante* le cas contraire.

Dès lors, on peut définir une *solution* comme étant une affectation totale consistante, c'est-à-dire une affectation d'une valeur à chaque variable sans violer aucune contrainte.
Il est à remarquer que, parfois, une affectation entraîne la disparition de toutes les valeurs d'un domaine : on dit alors qu'un échec se produit ; le dernier choix d'affectation est alors remis en cause, il y a un backtracking ou retour en arrière et une nouvelle affectation est tentée.

Enfin, autre qu'une solution unique, d'autres cas de figures peuvent exister concernant l'issu de la résolution d'un CSP, ce qui pose un problème d'optimisation supplémentaire à résoudre.

2.5.1.5 Optimisation d'un problème de satisfaction de contraintes

Il est courant pour un CSP d'avoir trop de contraintes pour qu'elles puissent être satisfaites, c'est le *CSP surcontraint*. Malgré l'absence d'une solution dans ce cas, ce problème peut être appréhendé selon deux approches qui tentent principalement de trouver une affectation totale répandant à certains critères.

La première approche s'agit de trouver l'affectation totale qui viole le moins de contraintes possibles. Le problème est appelé alors *max-CSP* dans la mesure où on cherche à maximiser le nombre de contraintes satisfaites.

La deuxième alternative propose d'affecter un poids à chaque contrainte selon son degré d'importance. Le résultat retenu est celui de l'affectation totale qui minimise la somme des poids des contraintes violées, c'est le principe de *CSP valué*.

De l'autre côté, il est très répandu d'avoir un *CSP souscontaint* disposant de plusieurs solutions. Dans ce cas, le mécanisme de rechercher une solution optimale passe par l'ajout d'une fonction de coût qui associe une valeur numérique à chaque solution selon la qualité de cette solution. Le but se résume désormais de trouver la solution CSP qui maximise cette fonction.

Il existe encore d'autres moyens permettant d'aider à aboutir à la solution optimale. On peut citer dans ce contexte, la fixation des stratégies de choix de variables et de valeurs avec la définition des critères permettant de choisir la prochaine variable et la prochaine valeur qui lui sera affectée.

Lorsque le problème à résoudre est trop grand, on peut recourir à des méthodes de décomposition qui vont permettre de décomposer le problème en plusieurs parties, les résoudre, si possible, d'une façon indépendante et enfin de les recombiner.

2.5.2 Planification globale de tâches dans un réseau TTEthernet

Comme déjà abordé dans la section précédente, l'extension INET implémentant le modèle TTEthernet fournit seulement les blocs fonctionnels responsables à faire tourner le protocole sans prévoir un outil de planification des messages TT, ce qui représente un vrai handicap au travail de simulation d'un protocole fondé sur le principe de déterminisme, où les nœuds doivent se maintenir synchronisés pour pouvoir suivre un plan de communication commun.

Cette partie aborde le problème de planification statique des messages TT tel qu'il est appréhendé par plusieurs recherches (Steiner, 2010; Suethanuwong, 2012; Tamas-Selicean, Pop et Steiner, 2012) tout en accordant une attention particulière au travail (Steiner, 2010)

qui a aidé à élaborer une spécification formelle de contraintes de planification pour les réseaux à déclenchement temporel à sauts multiples.

2.5.2.1 Présentation du problème

Contrairement à certains systèmes de communication à déclenchement temporel (exemple : FlexRay pour l'industrie automobiles, SAFEbus et TTP en avionique) dont le fonctionnement est basé sur la diffusion pour opérer sur le bus de communication ou une topologie en hub, la technologie Ethernet commuté a permis de placer TTEthernet comme premier protocole à déclenchement temporel supportant une communication concurrente de messages TT pour une topologie à sauts multiples.

Cette concurrence représente l'une des contraintes de base qui pèsent sur la planification des trames TT. Le problème se complique d'avantage avec l'augmentation de la taille de réseau et des tâches à traiter.

L'approche adoptée par le travail (Steiner, 2010) consiste à assigner des intervalles de temps à chaque participant à la communication. Cela instaure une exclusion mutuelle au profit des différents participants, qui vont se permettre, en synchronisant leurs horloges locales les unes aux autres, d'éviter tout conflit de ressources partagées durant le temps imparti à chacun.

Un autre facteur à prendre en considération est celui de types de trafics qui coexistent avec le trafic TT sur le même réseau (RC et/ou BE). Cela impose des contraintes supplémentaires à la planification qui doit, obligatoirement, prendre en compte les caractéristiques du trafic RC. De même le principe de précédence auquel bénéficie le trafic TT risque, en cas de planification très rapprochée des messages TT, de ne pas permettre aux messages RC d'être délivrés dans la limite de temps permise ainsi qu'entraver complètement le trafic BE.

Le scénario traitant le problème de criticité mixte de messages (TT+RC) a été étudié par (Tamas-Selicean, Pop et Steiner, 2012). Leur but était de générer une planification statique

hors ligne des messages TT de telle façon que les limites temporelles des messages TT et RC soient respectées tout en minimisant le délai de bout en bout des messages RC.

La recherche (Suethanuwong, 2012) s'est intéressée quant à elle, au problème de l'optimisation du trafic BE présent conjointement avec un trafic TT. Cela se fait à travers divers agencements des périodes allouées aux messages TT afin d'améliorer le flux BE sans nuire au comportement temporel vis-à-vis aux messages TT.

D'autres contraintes ont été examinées par l'article (Steiner, 2010). Elles varient entre l'imposition des règles de synchronisation en relation avec les liens de données, la spécification de la capacité mémoire des commutateurs et l'ordonnancement des applications en cas où il existe une interdépendance entre les tâches.

Une fois toutes les contraintes sont fixées, le problème de planification peut alors être formulé sous forme de systèmes de contraintes. La solution à ce système constitue le modèle de planification recherché : le système de contraintes sera formulé en logique de premier ordre. Par la suite, la résolution en sera faite en utilisant le solveur à usage général SMT. Celui-ci est un outil Yices équipé de procédures de décision qui vont résoudre ce système et générer le modèle de planification si le système est satisfiable.

L'utilisation de l'outil prêt à l'emploi SMT d'Yices pour synthétiser un plan de transmission en (Steiner, 2010) a permis au réseau TTEthernet de planifier des centaines d'instances de messages. Toujours dans ce contexte et afin de pouvoir améliorer la taille des réseaux TTEthernet traitable, une autre approche a été expérimentée en (Steiner, 2010). Elle consiste à implémenter un tt-ordonnanceur intégrant les fonctionnalités du solveur SMT Yices à l'aide de Yices C-API. Cela a presque permis de doubler le nombre d'instances de trames traitées.

2.5.2.2 Concepts et notions de bases

Il est important de pouvoir exprimer formellement la topologie d'un réseau TTEthernet afin de pouvoir spécifier avec pertinence les différentes contraintes d'un problème de planification. La figure 2.17 explicite la structure organisationnelle d'un réseau TTEthernet se basant sur les trois concepts : topologie physique, chemin de données et liens de données.

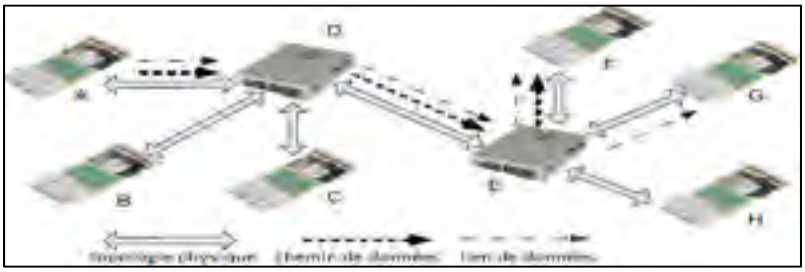

Figure 2.17 Organisation d'un réseau TTEthernet
Tirée de Steiner (2010)

Le réseau est représenté par un graphe non orienté G(V,E) où V est l'ensemble de sommets v_i regroupant les terminaux et les commutateurs, alors que E est l'ensemble des arêtes modélisant les lignes physiques connectant les différents sommets.

Un lien physique est bidirectionnel, chaque direction est interceptée par *un lien de données*. L'ensemble des liens de données est noté L, tel que :

$$\forall v_1, v_2 \in V : (v_1, v_2) \in E \Rightarrow [v_1, v_2] \in L, [v_2, v_1] \in L \tag{2.9}$$

Tirée de Steiner (2010)

Un chemin de données p est constitué d'une séquence de liens de données l_i reliant l'émetteur avec un seul récepteur :

$$p = [[v_1, v_2], \ldots, [v_{(r-1)}, v_r]] \tag{2.10}$$

Tirée de Steiner (2010)

La communication entre l'émetteur et le récepteur se fait à travers des trames. L'ensemble des trames f_i d'un chemin de données est noté *F*. Chaque lien de données [v_1, v_2] d'un chemin de données est caractérisé par une instance de trame notée $f_i^{[v1, v2]}$.

Chaque instance de trame $f_i^{[vk, vl]}$ est complètement définie par sa période, sa longueur et son offset. Les deux premiers doivent être connus au préalable, alors que l'offset est calculé et assigné par l'ordonnanceur :

$$f_i^{[vk, vl]} = \{f_i.periode, f_i^{[vk, vl]}.offset, f_i.length\} \quad (2.11)$$

Tirée de SAE (2011)

Enfin, l'ensemble de chemins de données reliant un émetteur à plusieurs récepteurs forme un *lien virtuel vl*, on le note formellement avec :

$$vl = U\, p_i \quad (2.12)$$

Tirée de Steiner (2010)

Un lien virtuel est un concept important pour un réseau TTEthernet vu qu'il assure la séparation spatiale entre les trames à criticité mixte. Il a, d'un point de vue structurelle, une structure arborescente où l'émetteur est la racine et est noté *first* (f_i), tandis que les récepteurs sont les feuilles et sont notés *last* (f_i). L'ensemble des lignes virtuels sont notés *VL*.

2.5.2.3 Spécification formelle des contraintes de planification TTEthernet

Comme présenté précédemment, il existe plusieurs contraintes modélisant le problème de planification. Le nombre de contraintes peut varier d'un système à un autre selon ses caractéristiques et les exigences à satisfaire.

Le travail (Steiner, 2010) aborde six contraintes potentielles d'un réseau TTEthernet, leur but est de définir les offsets de toutes les instances de trames d'un réseau TT.

2.5.2.3.1. Contrainte de non-conflits

Cette contrainte a pour but de vérifier que, à un instant t, chaque instance de trame dispose d'un accès exclusif à un lien de données. Cela permet de garantir qu'un nœud n'émet ou ne relaye une trame qu'après de s'assurer que la trame précédente a été complètement délivrée.

Trois cas de figure peuvent se dresser pour cette contrainte selon la nature des périodes de trames :

Le premier cas concerne le scénario où toutes les trames disposent d'une période unique pour transmettre. La gestion des conflits est exprimée formellement par la contrainte suivante :

$$\begin{aligned} &\forall\ [v_k, v_l] \in L,\ \forall f_i, f_j \in F : \\ &((f_i \neq f_j) \wedge \exists f_i^{[v_k, v_l]} \wedge \exists f_j^{[v_k, v_l]}) \Rightarrow \\ &((f_i^{[v_k, v_l]}.offset \geq f_j^{[v_k, v_l]}.offset + f_j.length) \\ &\vee (f_j^{[v_k, v_l]}.offset \geq f_i^{[v_k, v_l]}.offset + f_i.length)) \end{aligned} \tag{2.13}$$

Tirée de Steiner (2010)

Le deuxième scénario traite des trames ayant des périodes différentes, mais harmoniques. Un nouveau concept doit être ajouté dans ce cas, il s'agit de la constante *LCM* calculée en tant que le plus petit multiple commun de toutes les périodes harmoniques des trames. Il aide à avoir une hyper-période commune à toutes les trames. La contrainte ci-dessous vise à y positionner les instances de trames sans conflits.

$$
\begin{aligned}
&\forall\ [v_k\ ,\ v_l]\ \in L,\ \forall f_i\ ,f_j\ \in F,\\
&\forall a \in [0..\left(\frac{LCM(F.period)}{fi.period}-1\right)]\\
&\forall b \in [0..\left(\frac{LCM(F.period)}{fj.period}-1\right)]:\\
&((f_i \neq f_j) \wedge \exists f_i^{[v_k\ ,\ v_l]} \wedge \exists f_j^{[v_k\ ,\ v_l]}) \Rightarrow\\
&((a \times f_i.period) + f_i^{[v_k\ ,\ v_l]}.offset \geq\\
&(b \times f_j.period) + f_j^{[v_k\ ,\ v_l]}.offset + f_j.length) \vee\\
&((b \times f_j.period) + f_j^{[v_k\ ,\ v_l]}.offset \geq\\
&(a \times f_i.period) + f_i^{[v_k\ ,\ v_l]}.offset + f_i.length))
\end{aligned}
\tag{2.14}
$$

Tirée de Steiner (2010)

Le troisième scénario vise à simplifier la complexité de la planification en divisant l'hyper-période en slots de temps égaux. Le but est, désormais, d'assigner à chaque instance de trame le numéro du slot adéquat. C'est cette alternative qui a été adopté par (Steiner, 2010).

$$
\begin{aligned}
&\forall\ [v_k\ ,\ v_l]\ \in L,\ \forall f_i\ ,f_j\ \in F,\\
&\forall a \in [0..\left(\frac{LCM(F.period)}{fi.period}-1\right)]\\
&\forall b \in [0..\left(\frac{LCM(F.period)}{fj.period}-1\right)]:\\
&((f_i \neq f_j) \wedge \exists f_i^{[v_k\ ,\ v_l]} \wedge \exists f_j^{[v_k\ ,\ v_l]}) \Rightarrow\\
&((a \times f_i.period) + f_i^{[v_k\ ,\ v_l]}.offset \neq\\
&(b \times f_j.period) + f_j^{[v_k\ ,\ v_l]}.offset)
\end{aligned}
\tag{2.15}
$$

Tirée de Steiner (2010)

2.5.2.3.2. Contrainte de dépendance de chemins

Cette contrainte vise à garantir la synchronisation des temps d'envoi entre deux liens de données adjacents d'un chemin de données *p*. De ce fait, le lien de données [v_j, v_y] qui succède au lien [v_x, v_j] ne doit commencer l'envoi qu'après un temps suffisant à la réception

de l'instance de trame envoyée par son prédécesseur [v_x, v_j] soit *max (hopdelay)*, c'est-à-dire le temps maximal d'un saut entre deux nœuds.

$$\forall vl \in VL : \forall p_i \in vl : \forall\ [v_x , v_j], [v_j , v_y] \in p_i :$$
$$(f_i^{[v_j , v_y]}.offset) - (f_i^{[v_x , v_j]}.offset) \geq max(hopdelay) \quad (2.16)$$

Tirée de Steiner (2010)

2.5.2.3.3. Contrainte de mémoire limitée de commutateurs

Cette contrainte aide à fixer la taille mémoire des commutateurs en contrôlant le temps maximal qu'une instance de trame passe au niveau des commutateurs. Le temps d'attente d'une instance de trame est calculé à partir de la différence entre l'offset d'un lien de données avec l'offset de son prédécesseur. Étant donné que dans notre cas les offsets sont des numéros entiers des slots dans un hyper-cycle, le temps d'attente calculé va être un entier positif modélisant le nombre d'instances de trames figurant en même temps dans la mémoire. Ce nombre est borné par la constante *membound* qui représente le nombre maximal d'instances de trames autorisées dans la mémoire des commutateurs.

$$\forall vl \in VL : \forall p_i \in vl : \forall\ [v_x , v_j], [v_j , v_y] \in p_i :$$
$$(f_i^{[v_j , v_y]}.offset) - (f_i^{[v_x , v_j]}.offset) \leq membound \quad (2.17)$$

Tirée de Steiner (2010)

2.5.2.3.4. Contrainte de relai simultané

Il arrive parfois qu'une trame affectée à un lien virtuel donné diverge à partir d'un certain nœud vers des chemins de données différents. Cette contrainte a pour but d'assurer que les deux instances de la trame soient envoyées simultanément à partir du point de divergence.

$$\forall vl \in VL : \forall p_k, p_l \in vl :$$
$$\forall [v_j , v_b] \in p_k, \forall [v_j , v_d] \in p_l : \quad (2.18)$$
$$(f_i^{[v_j , v_b]}.offset) = (f_i^{[v_j , v_d]}.offset)$$

Tirée de Steiner (2010)

2.5.2.3.5. Contrainte du temps de bout en bout

Cette contrainte surveille le comportement déterministe d'un réseau TTEthernet. Comme son nom l'indique, son rôle est de s'assurer que le délai de transmission d'une trame ne dépasse pas une certaine latence maximale *max(latency)* qui est une constante caractéristique de chaque application

$$\forall f_i \in F : \forall [v_l , v_k] \in last(f_i) :$$
$$f_i^{[v_l , v_k]}.offset - f_i^{first(f_i)}.offset < max(latency) \quad (2.19)$$

Tirée de Steiner (2010)

2.5.2.3.6. Contrainte niveau application

Cette contrainte s'impose au cas où il y a une interdépendance entre les tâches des applications. C'est-à-dire si une trame f_b dépend d'une autre trame f_a alors f_a doit être envoyée au moins Δ temps avant la trame f_b.

$$first(f_a) = [v_l , v_k] \wedge first(f_b) = [v_m , v_n]:$$
$$f_a^{[v_l , v_k]}.offset + \Delta \leq f_b^{[v_m , v_n]}.offset \quad (2.20)$$

Tirée de Steiner (2010)

2.5.3 Discussion

L'utilisation la puissance de calcul et d'interprétation d'un solveur SMT d'Yices pour générer un plan de transmission constitue une grande aide pour les travaux de simulation des

modèles TTEthernet surtout avec la croissance et l'augmentation en complexité des systèmes simulés. Une puissance qui peut être améliorée d'avantage avec la possibilité d'intégrer ses fonctionnalités via l'API d'Yices.

2.6 Conclusion

A la fin de ce chapitre, nous avons présenté et discuter les différents principes et algorithmes régissant le fonctionnement du protocole TTEthernet tout en faisant une revue de littérature des principaux travaux qui ont marqué la simulation TTEthernet.

CHAPITRE 3

REALISATION D'UN MODELE DE SIMULATION TTETHERNET

3.1 Introduction

La réalisation d'un modèle de simulation TTEthernet passe essentiellement par deux étapes : la réalisation de la structure fonctionnelle du réseau TTEthernet et l'élaboration d'un plan de transmission régissant les trames TT du réseau.

Ce chapitre comprend deux sections :

La première section décrit l'élaboration de la structure d'un réseau TTEthernet implanté sur une partie du système de gestion de vol (FMS). On y présente les différents composants instanciés à partir du modèle TTEthernet basé sur l'extension d'INET d'OMNeT++ (CoRE4INET). On y aborde aussi les blocs fonctionnels internes de ces composants et leurs interactions.

La deuxième partie vient pour combler la lacune du modèle CoRE4INET en termes de mécanismes de planification des tâches TT. On y présente une adaptation du travail (Steiner, 2010) pour réaliser un plan de transmission des trames TT exploitable en simulation par le modèle réalisé.

3.2 Réalisation des blocs fonctionnels du modèle TTEthernet

Pour montrer un cas type d'utilisation du protocole TTEthernet, on l'a implanté sur un sous-système de gestion de vol (voir figure 3.1). Ce dernier gère et contrôle l'affichage d'informations de navigation à l'aide de l'interaction avec l'équipage via d'écrans et de claviers.

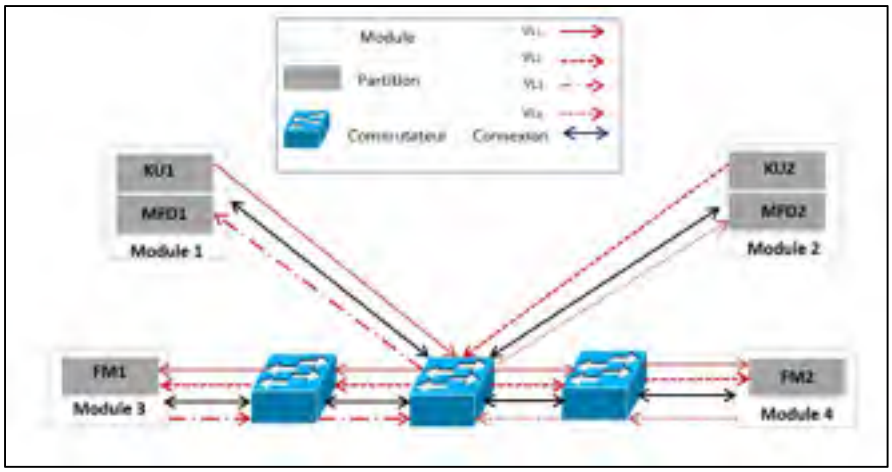

Figure 3.1 Structure d'un sous-système de gestion de vol

Pour des raisons de sûreté, le système repose sur une architecture redondante à double voies (i=1 pour le pilote, i=2 pour le copilote), où la fonction KU_i contrôlant le clavier et la fonction MFD_i gérant l'affichage de navigation sont liées au gestionnaire de vol FM_i.

Le fonctionnement de ce système peut être résumé, tel que présenté par (Lauer, 2012), comme suit :

1) La fonction KU_i reçoit une requête d'affichage d'un point de navigation entrée par le pilote i.
2) la fonction KU_i transmet la requête aux gestionnaires de vols FM_1 et FM_2
3) FM_1 et FM_2 interrogent en parallèle la base de données de navigation.
4) La base de données de navigation retourne les informations de navigation pertinentes aux FM.
5) Chaque FM_i envoie périodiquement, au MFDi correspondant, les informations à afficher.

Contrairement à la version originale du système FMS basé sur AFDX, l'utilisation du protocole TTEthernet pour connecter les différents modules du sous-système à réaliser va permettre d'introduire en plus du trafic RC remplaçant du trafic AFDX d'origine, les trafics TT et BE, et ce, selon les besoins de simulation.

3.2.1 Exigences temps réel

Pour des raisons de sûreté du sous-système FMS, le scénario de fonctionnement décrit précédemment exige la satisfaction de certaines exigences temps réel. Deux exigences, propres à notre scénario de fonctionnement, peuvent être résumées comme suit, telles que présentées par (Lauer, 2012) :

- **Exigence de latence relative à une chaine fonctionnelle :** elle garantit une réponse suffisamment rapide, par le système, à une requête. De ce fait, l'écran du pilote doit afficher les informations du prochain point de navigation au plus tard 700 ms après la saisie du pilote.
- **Exigence de cohérence sur chaines fonctionnelles divergentes :** elle garantit qu'un événement en entrée du système génère plusieurs événements arrivant dans une même fenêtre temporelle. Conformément à ce principe, les écrans du pilote et du copilote doivent afficher les informations du prochain point de navigation saisi par le pilote dans un intervalle de 500 ms.

3.2.2 Réalisation de la topologie

Selon la description précédente du sous-système de FMS à réaliser, la topologie contiendra essentiellement trois commutateurs et quatre terminaux.

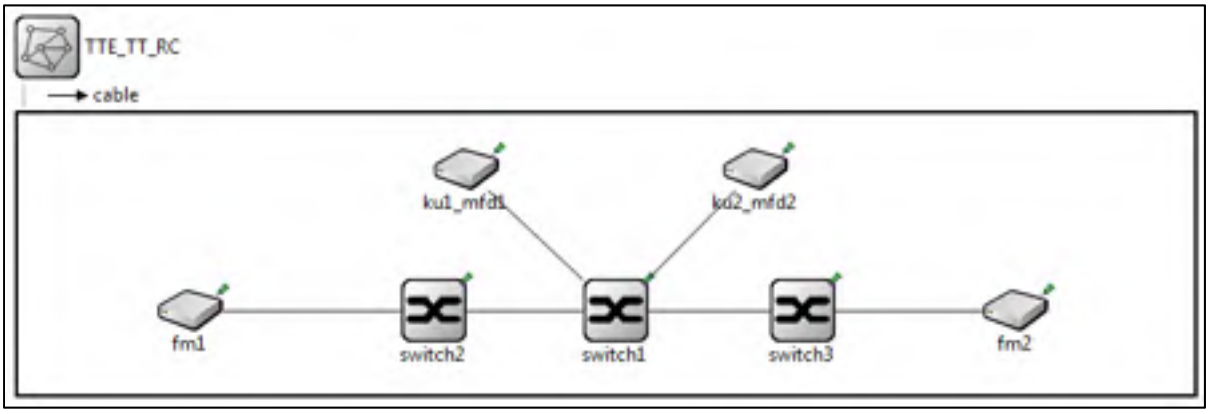

Figure 3.2 Topologie du sous-système de gestion de vol

Chaque module du sous-système de gestion de vol (figure 3.1) est assimilé à un terminal dans la topologie à réaliser. Par conséquent, un terminal peut regrouper plusieurs partitions, ce qui est le cas pour les terminaux ku1_mfd1 et ku2_mfd2. Ces derniers regroupent deux partitions qui exercent deux fonctions différentes. Cela se traduit effectivement par l'attribution, via des générateurs de trafics, d'une application intrinsèque à chaque partition. Chaque application est configurable selon le type de trafic qu'elle génère (TT, RC ou BE).

La séparation spatiale des données transmises par chaque partition du sous-système de gestion de vol est assurée par l'affectation d'un lien virtuel (VL) propre au trafic de chaque application. La séparation temporelle des trafics des différentes partitions est assurée grâce à la planification des tâches.

Pour des raisons de simulation, liées essentiellement à la validation du comportement du protocole TTEthernet, on va intervenir au niveau du nombre et de types de trames générées par chaque partition du sous-système : chaque partition doit pouvoir générer un exemplaire de chaque type de trafic (TT, RC et BE).

Cela nécessite de multiplier les générateurs de trafics pour chaque partition, pour atteindre trois en total, un pour chaque type de trafic. De même, le nombre de liens virtuels va devoir être dupliqué, vu qu'on a besoin, désormais, d'un lien virtuel pour chaque trafic TT et un autre pour son exemplaire en RC.

Le tableau suivant montre la répartition des liens virtuels pour le réseau TTEthernet interconnectant les différents modules du sous-système à réaliser.

Tableau 3.1 Répartition des liens virtuels pour le sous-système de gestion de vol

Lien virtuel	Source	Destination	Direction	Type de trafic
VL_1	KU_1	FM_1, FM_2	$\{S_1, S_2\}$, $\{S_1, S_3\}$	TT
VL_{11}	KU_1	FM_1, FM_2	$\{S_1, S_2\}$, $\{S_1, S_3\}$	RC
VL_2	KU_2	FM_1, FM_2	$\{S_1, S_2\}$, $\{S_1, S_3\}$	TT
VL_{22}	KU_2	FM_1, FM_2	$\{S_1, S_2\}$, $\{S_1, S_3\}$	RC
VL_3	FM_1	FMD_1	$\{S_2, S_1\}$	TT
VL_{31}	FM_1	FMD_1	$\{S_2, S_1\}$	RC
VL_4	FM_2	FMD_2	$\{S_3, S_1\}$	TT
VL_{41}	FM_2	FMD_2	$\{S_3, S_1\}$	RC

3.2.3 Modèle d'horloge

Compte tenu le débit du réseau TTEthernet qui est 100 Mbits/s, la durée d'un tick d'horloge est fixée à 80 ns. Cela a l'avantage de faire correspondre le temps de transmission d'un octet de données à un tick d'horloge, ce qui va faciliter, par la suite, le calcul et l'introduction d'autres paramètres de synchronisation y compris ceux définissant la planification des différentes tâches du réseau.

La déviation maximale par tick est de l'ordre de 20 ppm, soit 2 ps, alors que la variation moyenne d'horloge par cycle est définie par une distribution uniforme allant de -50 ps à 50 ps (*Uniform (-50ps, 50ps)*). Quant à la précision du système, elle est fixée à 500 ns.

3.2.4 Modèle du commutateur

La figure 3.3 montre les différents blocs fonctionnels du commutateur1 interconnectant le sous-système de gestion de vol. On en distingue essentiellement, les ports physiques, un module de synchronisation, un ordonnanceur, un module *beswitch*, des contrôleurs de trafics TT et RC, ainsi que des buffers spécialisés pour le trafic TT et d'autres pour le trafic RC.

Chaque port physique est une instance du module composé *TTEPHYPORT* regroupant un module de sortie appelé *shaper*, un module d'entrée *inControl* et un module *mac*.

Le module *shaper* s'intéresse aux trafics sortants, il les achemine vers le module *mac* selon leur ordre de priorité. L'autre sens de trafic est géré par le module *inControl* qui a pour rôle de vérifier la criticité des trafics venant du module mac en contrôlant le champ *CT-ID*. Selon ce dernier, il décide soit d'acheminer directement la trame vers l'un des contrôleurs de trafics *ctc*, s'il s'agit d'un trafic critique (TT ou RC), soit d'acheminer la trame vers le module *beswitch* assurant les fonctionnalités d'un commutateur Ethernet standard, dans le cas d'un trafic BE.

Le module mac assure une connexion point à point en mode full-duplex. Sa fonction est de transmettre le trafic sortant reçu du module *shaper* et d'acheminer les trames entrantes vers le module *inControl*.

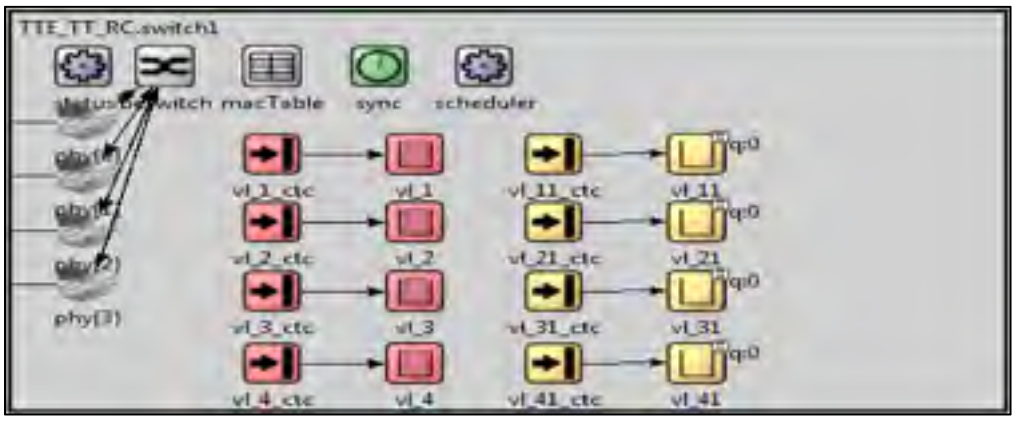

Figure 3.3 Blocs fonctionnels du commutateur1

La correction de l'horloge est assurée par *sync* instance du module *DummySync*, qui effectue cette tâche en se basant sur le temps de simulation, sur réception de l'évènement Sync Task Event de la part de l'ordonnanceur.

De son côté, l'ordonnanceur coordonne avec d'autres modules afin de maintenir la synchronisation et transmettre les trames TT selon un plan de transmission préétabli. Ce

fonctionnement est dirigé par plusieurs modèles d'évènements déclenchés via plusieurs messages, dont on cite :

- *SchedulerActionTimeEvent* : ce sont des messages de type *ACTION_TIME_EVENT*, ils sont utilisés pour les évènements de l'ordonnanceur qui se déclenchent dans un temps spécifique du cycle.
- *SchedulerTimerEvent* : ce sont des messages de type *TIMER_EVENT*, ils sont utilisés pour les évènements de l'ordonnanceur dont le déclenchement est lié à l'expiration d'un certain temps (*timer*).
- *SchedulerEvent* : ce sont des messages de type NEW_CYCLE utilisés au début de chaque nouveau cycle.

L'adaptation d'un commutateur avec le contexte de fonctionnement du sous-système passe par bien décrire les liens virtuels qu'il gère.

Dans le cas du commutateur 1, il faut gérer huit liens virtuels véhiculant les trames TT et RC des différents nœuds qu'il connecte et dont on doit spécifier, selon le type de trafic, les modules adéquats à associer pour chacun.

Dans le cas d'un trafic TT, il faut prévoir pour chaque lien virtuel un contrôleur de trafic *ctc*, instance du module *TTIncoming*. Ce contrôleur vérifie la conformité du trafic TT avec la fenêtre de réception qui lui est associée et œuvre à retarder la transmission d'une trame TT donnée jusqu'à son point de permanence (*permanence_pit*). Pour ce faire, il doit enregistrer un évènement TIME_ACTION_EVENT auprès de l'ordonnanceur via un message *SchedulerActionTimeEvent*.
Chaque contrôleur de trafic TT doit être obligatoirement lié à un double buffer instance de *TTDoubleBuffer* qui se charge de stocker les messages TT et de les envoyer, ensuite, en concordance avec une fenêtre d'envoi préconfigurée. Ce double buffer sauvegarde un seul message TT à la fois, qui sera remplacé à chaque arrivée d'un nouveau message.

Dans le cas d'un trafic RC, chaque lien virtuel doit être connecté à un contrôleur de trafic *ctc*, instance du module *RCIncoming*. Celui-ci vérifie la conformité du trafic RC avec un bag (bandwidth allocation gap) préconfiguré. En cas de concordance, le message sera envoyé vers une file d'attente instance du module *RCQueueBuffer* qui opère selon la politique *fifo* et s'assure d'envoyer les trames RC en respectant un *bag* prédéfini.

3.2.5 Modèle du terminal

Un terminal en modèle CoRE4INET se distingue d'un commutateur TTEthernet par la présence des composants gérant les relations avec la couche application et d'autres assurant la génération de trafics (voir figure 3.4)

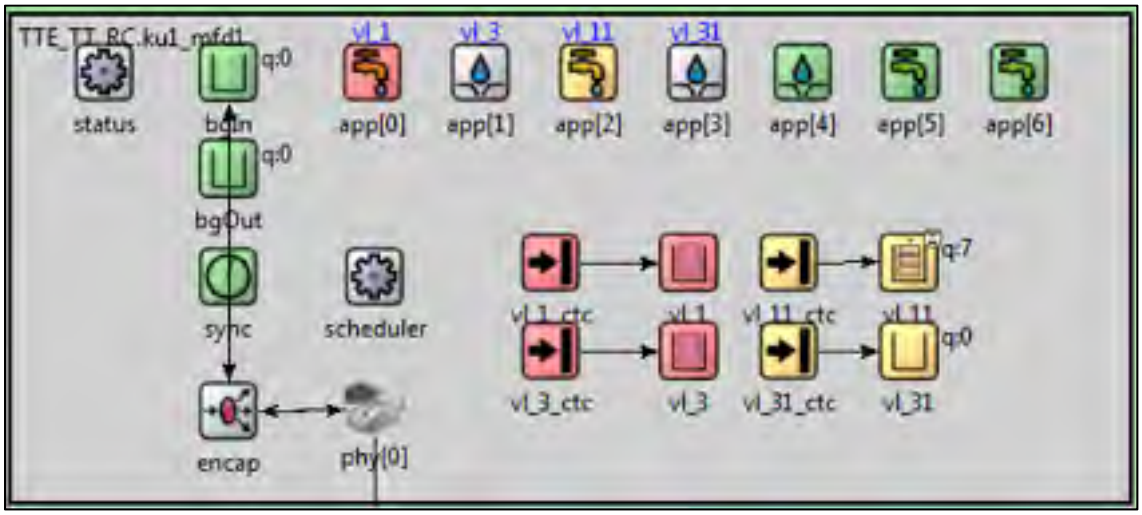

Figure 3.4 Blocs fonctionnels du terminal ku1_mfd1

Le composant *encap* (voir figure 3.4) instance du module *BGEtherEnCap* traite le trafic BE. Il effectue les tâches d'encapsulation/décapsulation des trames BE et veille à leur envoi vers l'application appropriées à l'aide de la valeur de leur champ *DSAP* (Destination Service Access Point).

Au niveau supérieur, on distingue deux types d'applications :

- **Les applications réceptrices de trafics** (TT ou RC) : elles sont instanciées à partir du module *CTTrafficSinkApp* et sont chargées de collectionner les statistiques concernant la transmission des trafics.
- **Les applications sources de trafics :** on en distingue deux variantes selon le type de trafic généré :
  - des instances de *TTTrafficSourceApp* : elles sont utilisées comme des générateurs de trafics TT où chaque trame TT est caractérisée par une période et un temps dans le cycle où elle sera transmise au buffer.
  - des instances du module *RCTrafficSourceApp* : elles fournissent des générateurs de trafics RC où les trames RC de chaque lien virtuel sont cadencées par un intervalle d'envoi.

Pour le terminal ku1_mfd1, on a besoin de définir quatre applications critiques, chacune d'entre elles est associée à un lien virtuel particulier : un générateur de trafic TT pour le lien virtuel vl_1, un générateur de trafic RC pour le lien virtuel vl_{11}, une application réceptrice de trafic TT venant du lien virtuel vl_3 et une dernière application réceptrice de trafic RC associée au lien virtuel vl_{31}.

D'autres applications doivent être associées au trafic BE pour l'envoi/réception des trames BE.

3.3 Élaboration d'un plan de transmission global pour les trames TT

La vocation première du protocole TTEthernet est de transmettre, suivant un plan de transmission global, un ensemble de tâches qui se partagent une base de temps commune.
Bien qu'elle ait bien défini l'ensemble des protocoles de synchronisation responsables à l'établissement et au maintien d'une base de temps commune, la spécification TTEthernet n'a pas défini d'approches de planification pour le trafic TT.

L'approche abordée par (Steiner, 2010) apporte une solution pour remédier à ce problème. Elle se base sur une spécification formelle prenant en considération toutes les contraintes influant la transmission des trames TT. Cependant, plusieurs adaptations sont nécessaires afin de pouvoir exploiter les résultats théoriques de l'approche au contexte pratique de la simulation.

3.3.1 Identification des besoins de la simulation en termes de planification

Après avoir réalisé les blocs fonctionnels de notre système, on est amené à remplir les paramètres de configuration pour chaque nœud (terminal et commutateur). Parmi ces paramètres il existe ceux qui sont dédiés à la planification des tâches TT. Ils représentent dans leur totalité (c'est-à-dire les paramètres de tous les nœuds) le plan de transmission du système.

La figure 3.5 montre les paramètres qui doivent être déduits à partir du plan de transmission pour le nœud ku1_mfd1.

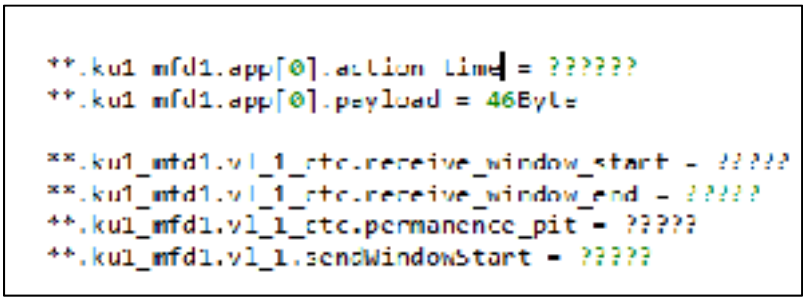

Figure 3.5 Les paramètres de configuration pour planification du nœud ku1_mfd1

Le paramètre *action_time* représente le temps d'envoi dans le cycle d'une trame TT, de l'application vers le buffer. Ce paramètre est propre aux terminaux générateurs de trames TT. De plus, la fenêtre de réception (*acceptance_window*) d'un message TT est délimitée par les deux paramètres *receive_window_start* et *receive_window_end*, alors que le temps d'attente

d'une trame TT avant son envoi vers le nœud suivant est fixé par *permanence_pit*. Enfin, le temps d'envoi de la trame TT doit être acquis à partir du paramètre *send_window_start*.

Cependant, l'approche (Steiner, 2010) évoque comme résultat un plan matérialisé par des périodes de temps (offsets) sous forme des indices entiers représentant l'ordre de chaque slot dans l'hypercycle.

Cela ne répond pas aux besoins de la simulation, qui requiert plutôt de déterminer un point précis dans le cycle, à partir duquel on peut déduire les autres paramètres. Ce point doit être, pour un nœud récepteur D, l'instant effectif d'envoi (*send_pit*) par un émetteur A d'une trame TT tel que présenté par la figure 3.6.

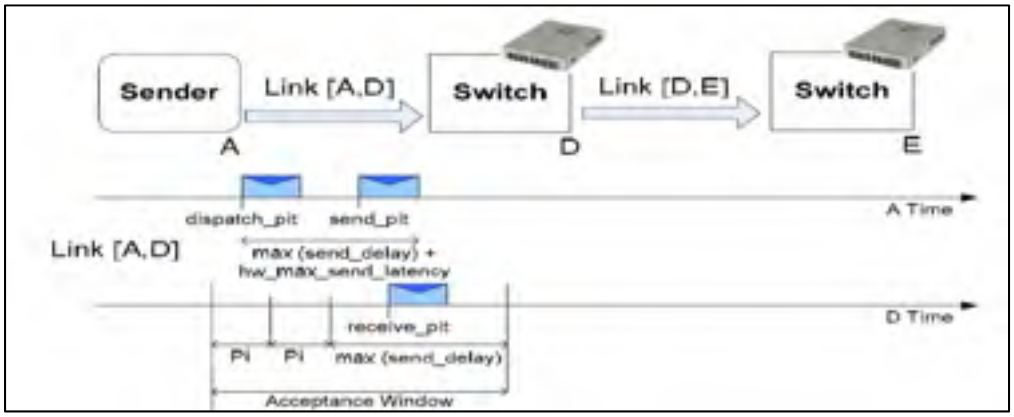

Figure 3.6 Les instants caractéristiques de la transmission d'une trame TT
Tirée de Steiner et al. (2012)

On a:

$$send\_pit \in [dispatch\_pit,\ dispatch\_pit + max(send\_delay) + hw\_max\_send\_latency]$$

où *dispatch_pit* est l'instant défini statiquement d'une période à partir de laquelle le transfert d'une trame est déclenché. Réellement, cet instant peut être retardé par certains traitements comme l'application d'une politique de gestion de flux particulière. Ce temps additionnel

qu'on appelle *max(send_delay)* peut être majoré d'avantage par un autre délai *hw_max_send_latency* imposé par le matériel autre que la latence due au stockage en buffer. La simulation fait abstraction du délai matériel, donc *hw_max_send_latency=0*. De plus, chaque trame TT reçue doit être relayée immédiatement vers le noeud du prochain saut (timely block). De ce fait, il n'y a pas de délais supplémentaires à l'envoi d'une trame TT d'où on a *send_pit = dispatch_pit* vu qu'en plus de *hw_max_send_latency, max(send_delay)=0*. Cela implique qu'il n'y a pas de temps d'attente en buffer, ce qui va être assimilé par le programme de simulation avec le paramètre *permanence_pit= -1 tick*. Il en découle aussi que *receive_window_end= send_window_start* vu qu'une trame TT est envoyée directement après sa réception.

Enfin, Il est important de noter que pour un terminal générateur de trames TT, la valeur de *send_pit* n'est autre que la valeur du paramètre *action_time* décrit précédemment.
De ce qui précède, on peut conclure pour les paramètres de configuration liés au plan de transmission, que pour un nœud donné du système on a:

- *action_time = send_pit* (en cas d'un terminal générateur de trames TT)
- *permanence_pit= -1 tick*
- *receive_window_end= send_window_start*

Les paramètres de la fenêtre de réception *receive_window_start* et *receive_window_end* sont calculés respectivement en se basant sur les équations (2.6) et (2.7) étudiées précédemment, une fois la valeur de *send_pit* est connue.

Dès lors, le plan de transmission pour la simulation est totalement déterminé une fois que la valeur de *send_pit* est déterminé pour chaque nœud impliqué dans la transmission.

3.3.2 Adaptation de la planification des tâches pour la simulation

Comme déjà montré, la planification des tâches passe par l'identification des instants effectifs de l'envoi des trames TT (*send_pit*).

La résolution du problème basée sur la spécification formelle des contraintes de la transmission TT peut être adaptée pour notre système et permet de rendre comme résultat les instants *send_pit* au lieu des numéros de slots de différentes offsets de l'hyper-cycle. Pour cela, quelques considérations pratiques de la simulation doivent être prises en compte.

3.3.2.1 Détermination de la limite temporelle des instants d'envoi

Le début d'un cycle de trames TT est toujours marqué par l'envoi de la trame PCF dont le rôle est d'assurer la synchronisation de l'ensemble des horloges du système. Par conséquent, l'envoi d'une trame TT dans un cycle doit se faire à un instant qui succède la fin de l'émission de celle-ci.
Une trame PCF comporte une charge utile de 46 octets et atteint une taille totale de 84 octets, ce qui correspond, avec un tick de 80 ns, à une marge minimale de 6,8 us avant l'envoi d'une trame TT par un terminal dans un cycle donné. Cette condition peut être traduite par l'ajout de la contrainte suivante :

$$\begin{aligned} &\forall\, first\,(f_i) = [v_l, v_k] \\ &f_i^{[v_l, v_k]}.send\_pit > pcf\_length \end{aligned} \tag{3.1}$$

où *first* (f_i) désigne le premier lien de données du lien virtuel. En cas d'une configuration redondante pour un système tolérant aux fautes, *first* (f_i) identifie les premiers liens de données de l'ensemble de liens virtuels redondants.

Pour notre système cette contrainte concerne les trois terminaux ku1_mfd1, ku2 et fm1.

3.3.2.2 Adaptation de la contrainte de non-conflits

Il existe trois alternatives pour cette contrainte telles que présentées précédemment. Dans le cadre de notre simulation et contrairement à (Steiner, 2010) qui a choisi la troisième alternative (2.15), nous allons opter pour la première, correspondant à l'équation (2.13). Cela revient au fait que travailler avec une période commune à toutes les applications permet d'apporter à la simulation une meilleure visibilité en termes de trames TT circulant via l'ensemble du réseau. En plus, cela facilite le référencement des instants d'envoi des trames les unes par rapport aux autres.

De plus les *send_pits* doivent, désormais, être espacés d'un délai minimum DME_{TT} séparant l'envoi de deux trames TT successives.

La contrainte adaptée peut être réécrite comme suit :

$$\begin{aligned}
&\forall\ [v_k\ ,\ v_l] \in L,\ \forall f_i, f_j \in F : \\
&((f_i \neq f_j) \wedge \exists f_i^{[v_k\ ,\ v_l]} \wedge \exists f_j^{[v_k\ ,\ v_l]}) \Rightarrow \\
&((f_i^{[v_k\ ,\ v_l]}.send_pit \geq f_j^{[v_k\ ,\ v_l]}.send_pit + f_j.DME_{TT}) \\
&\vee (f_j^{[v_k\ ,\ v_l]}.send_pit \geq f_i^{[v_k\ ,\ v_l]}.send_pit + f_i.\ DME_{TT}))
\end{aligned} \tag{3.2}$$

Cette contrainte est la plus gourmande en matière de lignes de code puisqu'elle doit être utilisée pour vérifier deux à deux l'ensemble des instances de trames TT qui se partagent un lien de données afin de garantir l'exclusivité d'accès à cette ressource commune.

3.3.2.3 Autres contraintes

On conserve les contraintes (2.16) à (2.20) qui demeurent valides en raisonnant en termes de *send_pit* au lieu d'*offsets*.

3.3.2.4 Génération des résultats

La traduction de la spécification formelle de l'ensemble des contraintes en langage Yices permet de donner, après exécution, le tableau 3.2

Les résultats ci-dessous sont obtenus pour une planification des trames TT de taille totale 750 octets du sous-système de gestion de vol avec un délai de traitement négligeable au niveau des FMi. La configuration de l'ensemble des nœuds du réseau en accord avec ces valeurs permet de garantir le respect de l'ensemble des contraintes discutées précédemment.

Tableau 3.2 Les différents points d'envoi (*send_pit*) du sous-système de gestion de vol

	Trame f_a (vl_1)	**Trame f_b (vl_2)**	**Trame f_c (vl_3)**	**Trame f_d (vl_4)**
Ku1_mfd1	1236 ticks (98,88 us)	–	6410 ticks (512,8 us)	–
Ku2_mfd2	_	85 ticks (6,8 us)	–	5259 ticks (420,72 us)
Switch1	1989 ticks (159,12 us)	838 ticks (67,04 us)	5657 ticks (452,56 us)	4506 ticks (360,48 us)
Switch2	2742 ticks (219,36 us)	1591 ticks (127,28 us)	4904 ticks (392,32 us)	–
Switch3	2742 ticks (219,36 us)	1591 ticks (127,28 us)	–	3753 ticks (300,24 us)
Fm1	3495 ticks (279,6 us)	2344 ticks (187,52 us)	4151 ticks (332,08 us)	–
Fm2	3495 ticks (279,6 us)	2344 ticks (187,52 us)	–	3000 ticks (240 us)

3.4 Conclusion

Dans ce chapitre nous avons abordé les différentes étapes requises pour la réalisation d'une simulation fonctionnelle pour un réseau TTEthernet. Une présentation des résultats et une étude de cas seront présentées au dernier chapitre.

CHAPITRE 4

EXPERIMENTATION ET RESULTATS

4.1 Introduction

Après avoir réalisé un modèle simulable du réseau TTEthernet à travers l'exemple du sous-système de gestion de vol, ce chapitre se focalise initialement sur la validation du comportement de ce modèle via la vérification de l'adéquation des résultats obtenus avec l'ensemble des spécifications de TTEthernet.

La deuxième partie vise à mettre en valeur l'importance du processus de planification de tâches en TTEthernet à travers une étude de cas montrant l'impact de l'agencement des instants d'envoi (*send_pit*) sur la transmission des différentes classes de trafics en TTEthernet.

4.2 Environnement de travail

L'ensemble des simulations présentées dans ce chapitre ont été réalisées sur un ordinateur de processeurs Core i5 avec une puissance 2.27 GHZ et une mémoire RAM de taille 8 GO.

La construction et la manipulation de modèles se sont basées sur l'extension CoRE4INET du framework INET 2.4 s'exécutant sur l'environnement de simulation OMNeT++ 4.4.1.

On a également fait recours au langage de programmation et aux solveurs de théories de l'outil Yices 2 afin de traduire les spécifications formelles des contraintes de transmission TTEthernet en un programme exécutable permettant de vérifier la satisfiabilité de l'ensemble de contraintes et fournir le modèle de solutions correspondant, si c'est satisfiable.

4.3 Validation du comportement du système

Cette section s'intéresse à valider le comportement du système selon trois termes : la gestion de flux, la distribution de trafics et la correction d'horloge.

4.3.1 Gestion de flux

L'étude de gestion de flux sera faite via l'analyse de la variation, au cours du temps, des délais de bout en bout des trames de taille minimale et maximale de chaque type de trafics et ce, dans un réseau à forte charge.

4.3.1.1 Configuration et méthodologie

Pour réaliser cette simulation, les nœuds ku1_mfd1, ku2 et fm1 sont configurés pour envoyer chacun trois exemplaires de trames, une pour chaque type de trafic, à chaque cycle. Ces trames sont soit de taille minimale (46 octets de charge utile), soit de taille maximale (1500 octets de charge utile).

Il faut signaler ici l'importance du commutateur 1 qui est le point de liaison entre les différents nœuds du réseau. La charge qu'on y exerce détermine la charge effective du système.

Dans le but de garantir une charge maximale au réseau, on a stressé le commutateur 1 avec jusqu'à 100% de charge BE supplémentaire provenant du nœud fm2.

Ce test se déroule avec un cycle de 550 µs durant un temps de simulation de 550 ms. Les délais de bout en bout de plus de 10000 trames sont enregistrés dont la figure 4.1 présente ceux concernant le nœud fm1.

4.3.1.2 Résultat

La figure 4.1 résume les variations de délais de bouts en bouts pour les trames reçues au niveau du nœud fm1.

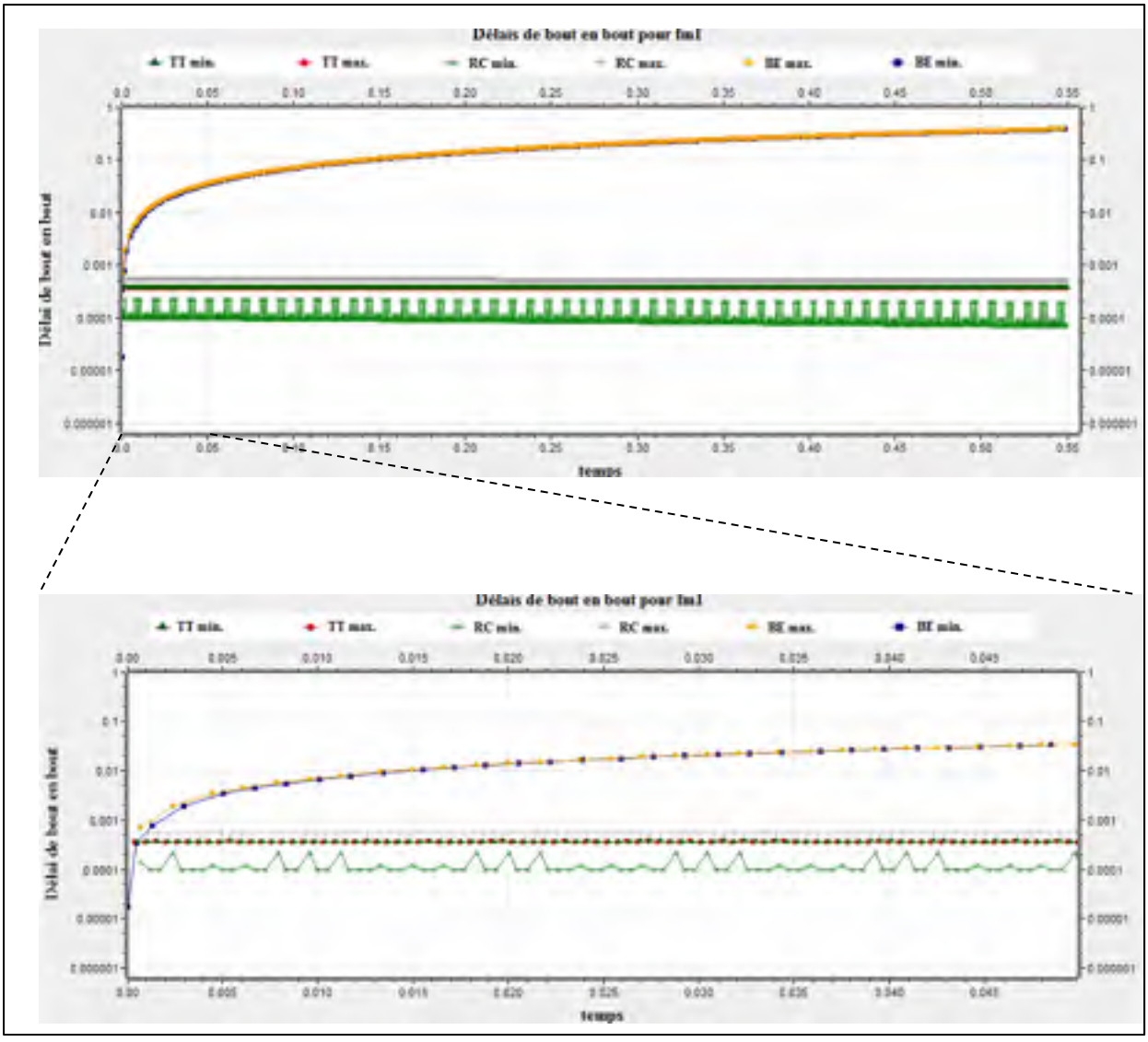

Figure 4.1 Variation de délais de bout en bout pour le nœud fm1

Conformément à la spécification TTEthernet, le système affiche un délai de bout en bout uniforme pour le trafic TT indépendamment de la taille de trames, dans un lieu à forte

charge. Ceci s'explique par plusieurs faits tels que l'aspect synchronisé du protocole, la gestion de priorités et le mécanisme de réservation de la ligne de transmission.

Le trafic RC affiche un faible délai de bout en bout qui varie selon la taille de la trame. Ce délai est stable pour les trames RC de taille maximale, mais présente plus de sensibilité au niveau des trames de taille minimale en montrant des légères fluctuations périodiques. Cela est relié essentiellement au mécanisme de réservation de ressources qui accorde la précédence aux trames TT et peut être influencé par conséquent par l'accumulation de la déviation d'horloge qui sera plus perceptible à un certain moment lors de l'allocation consécutive de petites espaces de transmission dans réseau à charge maximale.

Enfin, le trafic BE ne présente aucune garantie de livraison. Sa faible priorité et la monopolisation rapide des ressources par les trafics plus prioritaires expliquent l'augmentation brusque, dès le départ, des délais de bout en bout de ses trames, indépendamment de leurs tailles. Dès la saturation du réseau, les trames BE en excès seront perdues.

4.3.2 Distribution de flux

Cette expérience vise à étudier la distribution des trames de différents types de trafics lors de la réception à travers l'élaboration d'un histogramme des délais de bout en bouts des différents flux.

4.3.2.1 Configuration et méthodologie

Dans le cadre de cette simulation, chaque nœud doit envoyer trois exemplaires de trafics, une pour chaque type. La charge cette fois-ci est variable avec des tailles de trames aléatoires allant de 46 à 1500 octets de charge utile. On s'intéresse à la fin de simulation des délais de bout en bout des différentes trames de ku2 reçues par le nœud fm1.

4.3.2.2 Résultat

L'histogramme présenté par la figure 4.2 résume la distribution de délais de bout en bout des trames de ku2 au niveau du nœud fm1.

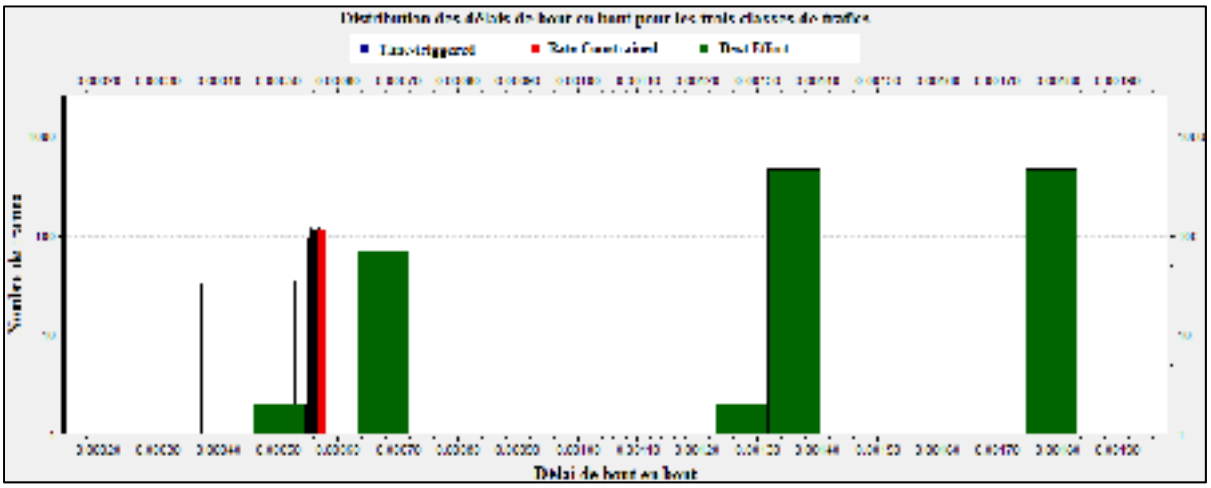

Figure 4.2 Distribution des délais de trames de ku2 à la réception en fm1

Cet histogramme confirme les délais de bout en bout faibles et constants pour les trames TT quel que soit la taille de trames et la charge du réseau. En revanche, ces délais sont variables, mais limités pour le trafic RC restant au-dessous de 600 µs. Enfin, pour le trafic BE, les délais sont beaucoup plus variables. Ceci revient à l'utilisation des files d'attentes et à la réservation de la bande passante par les trafics plus prioritaires.

4.3.3 Correction de l'horloge

La correction de l'horloge est un processus orchestré par l'ordonnanceur d'un nœud donné et vise à préserver l'état synchronisé avec l'ensemble du système et ce, périodiquement, pendant toute la période de fonctionnement.

La figure 4.3 décrit le comportement de ce processus pour le commutateur1.

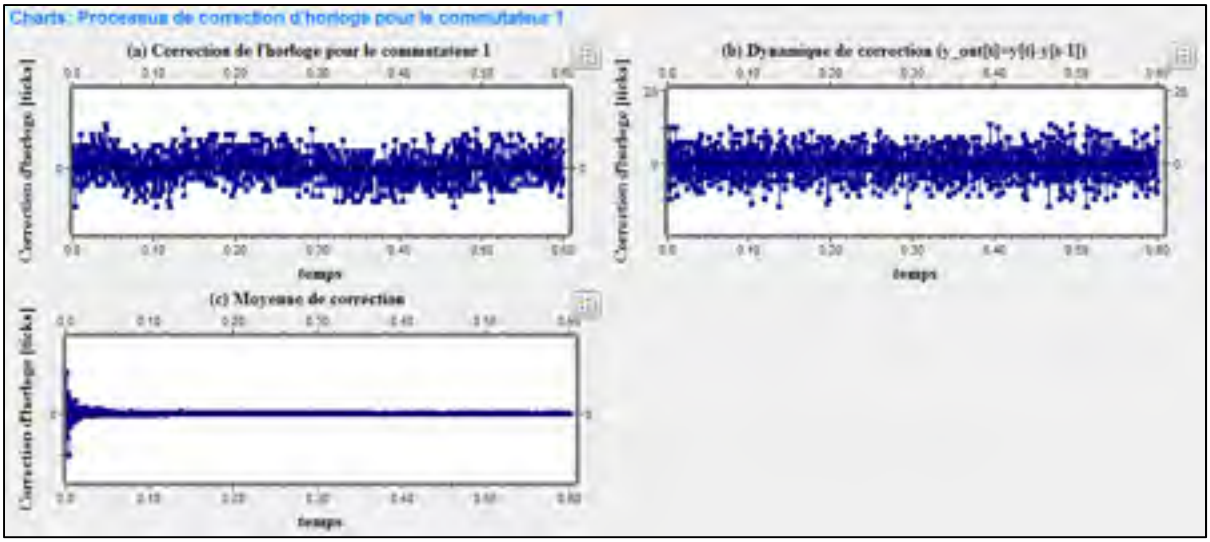

Figure 4.3 Processus de correction d'horloge pour le commutateur 1

La figure (a) présente la correction en temps réel de l'horloge du commutateur1, en nombre de ticks. Pour avoir une idée plus claire sur la dynamique de réaction du processus, la figure (b) dresse une courbe de la réponse du système à la déviation de l'horloge de l'instant précédant. Cela montre qu'à chaque déviation est associée une réponse compensatoire dans le sens opposé. L'ensemble de ces réactions s'affiche avec une allure symétrique à l'axe temporel.

L'impact de l'ensemble des opérations de correction appliquées à l'horloge locale du commutateur 1 peut être visualisé par le calcul de la moyenne des différentes corrections. Ceci donne une résultante nulle pour presque toute la période de simulation, ce qui montre l'efficacité du processus de correction vu que l'horloge a réussi à se maintenir synchronisé avec le système.

4.4 Etude de cas : Impact de l'agencement des périodes d'envoi des trames TT sur la transmission de l'ensemble des trafics TTEthernet.

Cette étude de cas vise à évaluer l'effet de l'agencement des points d'envoi (*send_pits*) des trames TT dans un cycle de transmission sur l'ensemble des trafics circulant sur le système.

Cette évaluation supervisera l'efficacité de transmission des différents types de trafics ainsi que leurs délais de transmission. Ceci revêt une importance capitale pour le système car même si le protocole est complètement favorable à la transmission des trames TT, tel qu'on a pu déduire par les expériences précédentes, il existe un autre trafic critique qui est le trafic RC, dont la performance reste tributaire de la gestion du trafic TT à cause de l'ordre de priorités instauré par le protocole TTEthernet. Une mauvaise performance pour les trames RC peut compromettre le système, d'où la nécessité de vérifier l'accommodation du trafic RC et de lui fournir un minimum de garantie.

Il existe deux cas possibles pour l'agencement des *send_pits* pour les trames TT : le premier est l'agencement contigu des *send_pits* alors que le deuxième est l'agencement distribué des *send_pits*.

4.4.1 Configuration et méthodologie

Pour les deux études de cas, on va conserver la même forme de trafics qui consiste à envoyer à chaque cycle de 750 µs, par les nœuds ku1_mfd1, ku2, et fm1 les flux suivants :

- Une trame TT de taille maximale.
- Deux trames RC de taille maximale, espacées par un intervalle d'envoi de 375 µs et ayant un *bag* de 150 µs.
- Deux trames BE de tailles variables

A chaque étude de cas est dédié, naturellement, un plan de transmission spécifique qui se charge de l'agencement des instants d'envoi des trames TT selon les exigences de chaque cas.

Les statistiques concernant l'efficacité de transmission de chaque type de trafics ainsi que leurs délais de bout en bout seront collectées à la fin de la simulation.

4.4.2 Premier cas d'utilisation : Agencement contigu des send_pits

Ce cas de figure existe si, lors d'un cycle de transmission, les envois des trames TT se succèdent sans délai d'attente, de telle façon que la fin de l'envoi d'une trame TT marque le début d'envoi d'une autre. La figure 4.4 illustre ce scénario.

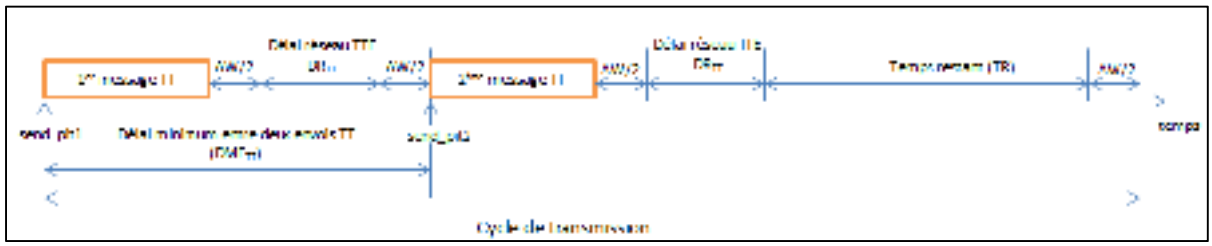

Figure 4.4 Scénario avec points d'envoi contigus pour deux trames TT

La figure 4.4 nous fait introduire un paramètre crucial pour la planification TT qui est le délai minimum entre l'envoi de deux trames TT (DME_{TT}) :

$$DME_{TT} = DT_{TT} + D_{AW} + DR_{TT} \tag{4.1}$$

où DT_{TT} est le délai de la trame TT qui varie selon sa longueur, D_{AW} est le délai de la fenêtre d'acceptance qui est équivalent dans notre cas à $2 * précision$ et DR_{TT} qui représente le délai réseau qu'on a fixé à 240 ns.

La planification dédiée à notre cas utilise le paramètre DME_{TT} à deux niveaux :

Au premier niveau, DME_{TT} figure dans la contrainte modifiée (3.2) pour fixer une distance minimale entre les trames TT. Ceci assure leur séparation temporelle lors du passage par le lien de transmission et garantit par conséquent la propriété d'exclusion mutuelle des ressources visée par cette contrainte.

Le deuxième endroit où DME_{TT} est utile, est la contrainte niveau application (2.20). C'est en attribuant à son paramètre Δ la valeur de DME_{TT} qu'on impose aux applications l'envoi des trames TT espacées exactement par DME_{TT}.

Les résultats obtenus à partir d'une période de simulation d'une seconde sont présentées par les tableaux suivants :

Tableau 4.1 Efficacités de transmission pour des instants d'envoi contigus

	Trames envoyées			**Trames reçues**			**Efficacité**
	vl1/vl11	**vl2/vl21**	**Total**	**vl1/vl11**	**vl2/vl21**	**Total**	
Trafic TT	1334	1334	2668	1333	1333	2666	99,92%
Trafic RC	2666	2666	5332	2664	2665	5329	99,94%
Trafic BE	5334			3			00,05%

Tableau 4.2 Délais de bout en bout pour des instants d'envoi contigus

	Délai de bout en bout minimal	**Délai de bout en bout maximal**
Trafic TT	368,83 µs	370,06 µs
Trafic RC	378,55 µs	966,37 µs
Trafic BE	146,4 µs	359,82 ms

Le premier cas d'utilisation affirme que l'agencement contigu des instants d'envoi des trames TT est favorable pour gérer la transmission pour notre système. Il offre une pleine efficacité pour les trafics critiques TT et RC tout en préservant le déterminisme temporel des trames TT avec des délais de bout en bout quasiment constants grâce à une variation très faible de 1,23 µs. La précédence des trames TT impose des délais de bout plus grands pour les trames RC avec une variation de 587,82 µs mais qui reste acceptable pour assurer une transmission en temps réel.

Le trafic BE a été bridé en termes d'efficacité et affiche un délai de bout en bout très variables. Ceci revient au fait que durant les cycles de transmission la voie de communication a été monopolisée complètement par les trafics critiques.

4.4.3 Deuxième cas d'utilisation: Agencement distribué des send_pits

Ce cas d'utilisation se présente lorsque le délai entre deux points d'envoi successifs de trames TT dépasse le délai d'envoi minimal DME_{TT} tel que présenté par la figure 4.5

Figure 4.5 Scénario avec points d'envoi distribués pour deux trames TT

Ce scénario illustre la séparation de deux points d'envoi (*send_pits*) successifs par un délai d'attente Δ qui est la somme de DME_{TT} avec un temps restant TR_{TT} :

$$\Delta = DME_{TT} + TR_{TT} + AW \tag{4.2}$$

De point de vue planification, ce paramètre va nous servir pour fixer la contrainte niveau application (2.20) afin d'imposer un délai Δ entre les instants d'envoi des différentes trames TT de chaque application.

Pour notre cas nous avons attribué un délai de 80 µs pour TR_{TT} soit dans notre contexte un délai de trame de taille 1000 octets.

Les résultats de simulation pour une durée d'une seconde sont illustrés comme suit :

Tableau 4.3 Efficacités de transmission pour des instants d'envoi distribués

	Trames envoyées			Trames reçues			Efficacité
	vl1/vl11	vl2/vl21	Total	vl1/vl11	vl2/vl21	Total	
Trafic TT	1334	1334	2668	1333	1333	2666	99,92%
Trafic RC	2666	2666	5332	1999	1999	3998	74,98%
Trafic BE	5334			1334			25%

Tableau 4.4 Délais de bout en bout pour des instants d'envoi distribués

	Délai de bout en bout minimal	Délai de bout en bout maximal
Trafic TT	368,89 µs	370,12 µs
Trafic RC	368,63 µs	250,38 ms
Trafic BE	146,4 µs	399,64 µs

Malgré la stabilité des performances du trafic TT, le deuxième cas d'utilisation a connu une baisse d'efficacité jusqu'à 25%, soit une perte totale de 1334 de trames RC par seconde ce qui est intolérable pour les systèmes critiques.

Cette baisse d'efficacité s'est accompagnée avec une croissance massive de la variabilité des délais de bout en bout pour atteindre une borne maximale de 250,38 ms, ce qui risque de dépasser le délai de 700 ms présenté précédemment comme exigence de latence relative à une chaine fonctionnelle et ce, en additionnant le temps de traitement encouru par les terminaux Fmi pour répondre à une requête (interaction avec la base de données et traitement local des données)

En revanche, le trafic BE a profité de la baisse des performances du trafic RC pour améliorer son efficacité qui a augmenté de 25%. Son rendement du point de vue délai de bout en bout s'est nettement amélioré en devenant beaucoup moins variable.

4.4.4 Discussion des résultats

L'étude de cas prouve que l'agencement des points d'envoi des trames TT est indispensable pour garantir le bon fonctionnement d'un système critique en temps réel.

Les bonnes performances du premier cas d'utilisation s'expliquent par la bonne gestion des instants d'envoi des trames TT qui a évité le gaspillage des ressources réseau en les tassant ensemble séparés par le délai minimum d'envoi DME_{TT}, pendant la première partie du cycle de transmission. Ceci a permis d'exploiter la totalité du temps restant *TR* pour transmettre le trafic RC avec une utilisation optimale de ressources ce qui s'est reflété positivement sur l'efficacité de transmission pour les trames RC et leurs délais de bout en bout.

Contrairement au premier scénario, les mauvaises performances du deuxième cas d'utilisation sont dues à une mauvaise gestion du temps restant TR_{TT}. Pour illustrer ce cas nous avons choisi d'attribuer un délai de 80 µs à TR_{TT}, ce qui est insuffisant pour transmettre une trame de taille maximale (pour cela il faut au moins 125 µs). Ce choix a causé le gaspillage de 80 µs par cycle de transmission vu qu'une trame RC ne peut pas y passer. Il a été exploité alors par le trafic BE.

L'autre conséquence de ce choix est, qu'après l'envoi de la dernière trame TT du cycle courant, le temps restant final *TRF*, contrairement au premier cas d'utilisation, n'est plus satisfaisant pour transmettre sans encombrement deux trames RC avant la date d'envoi de la première trame TT du cycle suivant. De ce fait, les trames RC vont s'accumuler progressivement dans le buffer RC causant une augmentation considérable de délais de bout en bout à cause du coût supplémentaire engendré par la gestion de mémoires de stockage.

L'ensemble de trames RC accumulées dans le buffer RC et qui reste non transmises à la fin de la période de simulation seront perdues.

4.5 Limites et perspectives

Malgré qu'on a parfaitement réussi à utiliser les modèles de CoRE4INET, l'extension d'INET pour l'environnement OMNeT++, pour modéliser le réseau TTEthernet et qu'on a pu, par la même occasion, adapter l'approche de planification des tâches évoquée par (Steiner, 2010) pour nos fins de simulation, des améliorations peuvent encore y être apportées.
En effet, le modèle TTEthernet proposé par CoRE4INET manque un module d'injection de fautes, sa réalisation est indispensable pour pouvoir tester la tolérance aux fautes des systèmes critiques, en plus de faciliter l'implémentation des modèles de simulation précis avec un spectre de scénarios simulables plus large.

De plus, notre code du programme généré en langage Yices, suite à l'application de l'approche de planification des tâches dépend naturellement de la topologie du système modélisé. Il croit exponentiellement et peut atteindre plusieurs centaines de lignes de codes avec des systèmes plus complexes. Il serait très utile par conséquent, d'ajouter un nouveau module capable de déduire la topologie du système et l'utiliser conjointement avec l'API Yices pour générer automatiquement le code de planification, ce qui permet, par la suite, au simulateur de configurer directement les ordonnanceurs impliqués dans le processus de gestion des tâches.

Enfin, la grande dépendance des performances de la transmission TTEthernet avec l'agencement des instants d'envoi des trames TT, telle que a été démontrée par notre étude de cas, nous fait parvenir à la conclusion qu'en cas de réseaux à criticité mixte où les deux trafics TT er RC coexistent, il est plus judicieux de réviser la procédure entreprise pour la planification des tâches dans le sens d'adapter les contraintes existantes aux nouvelles exigences de cette intégration. D'autres contraintes peuvent s'ajouter selon les prérequis du système ciblé. Les travaux de recherche tels que (Luxi et al., 2014; Zhao, 2014) prouve cette conclusion.

4.6 Conclusion

Ce dernier chapitre a servi de témoin pour valider le comportement du protocole TTEthernet régissant notre système à l'aide d'un ensemble d'expérimentations visant la gestion et la distribution des flux, ainsi que le processus de correction d'horloge.

On y illustre également, via une étude de cas, l'impact de la planification des points d'envoi des trames TT sur l'ensemble des performances du réseau TTEthernet.

CONCLUSION

Ce mémoire a été consacré à l'étude et la simulation du protocole TTEthernet en utilisant les modèles du projet CoRE4INET réalisés via l'extension du framework INET d'OMNeT++. Au cours de ce travail on a exploité les modèles CoRE4INET pour simuler le protocole TTEthernet sur un sous-système de gestion de vol (FMS). Les résultats de simulations ont permis de valider le fonctionnement du protocole en matière de délais de bout en bout, distribution de flux et correction d'horloge.

L'étude du protocole TTEthernet nous permet de conclure à une interdépendance de ses deux algorithmes de synchronisation et de tolérance aux fautes. En effet, le maintien de la synchronisation du système dépend étroitement du nombre, de la nature et du comportement des nœuds défaillants. Des modèles de fautes ont été alors définis pour le réseau TTEthernet où l'algorithme de tolérance aux fautes joue un rôle primordial grâce à ses fonctions *Central Guardian* et *High-Integrity* permettant de maitriser le comportement arbitraire des nouds défaillants via l'ajout des procédures de vérification et de contrôle de fonctionnement.

De même, l'analyse des différentes politiques de gestion de flux montre la primauté du blocage en temps opportun (*Timely Block*) sur les deux autres politiques. Cela revient essentiellement au taux de perte élevé engendré par la méthode avec préemption et le délai d'attente supplémentaire pouvant être exigé par la méthode avec brassage. Les solutions proposées pour remédier aux lacunes de ces deux dernières ne semblent pas très efficaces pour garantir la qualité hautement déterministe au réseau TTEthernet. En revanche, une bonne planification des tâches peut alléger énormément l'impact de toutes les politiques en considération.

L'étude comparative des deux modèles TTEthernet existants en revue de littérature nous a montré une similitude du point de vue des blocs fonctionnels. Malgré le fait que le modèle basé OPNET se distingue par un module sophistiqué d'injection de fautes, il semble

beaucoup moins mature que celui du projet CoRE4INET vu qu'à l'heure actuelle seuls les modèles de ce dernier sont publiés et expérimentés.

Malgré l'importance de la planification des tâches pour le protocole TTEthernet dont la vocation première est de transmettre, suivant un plan de transmission global, un ensemble de trames TT qui se partagent une base de temps commune, CoRE4INET se contente de définir les blocs fonctionnels d'un réseau TTEthernet et ne fournit pas un outil de planification. Pour pallier à ce problème, on a proposé dans le contexte de cette recherche une adaptation d'une approche de planification basée sur la spécification formelle des contraintes du réseau TTEthernet qui nous a permis de définir avec précision les instants d'envoi (send_pits) des différentes instances de trames TT et ce, en répondant à un ensemble d'exigences telles que l'exclusion mutuelle de l'accès aux ressources communes, la synchronisation entre les différents nœuds du réseau, la détermination de la taille des buffers pour les commutateurs et la fixation des délais de bout en bout.

La traduction de la spécification formelle des contraintes en programme exécutable s'est fait en concordance avec les paradigmes de la programmation par contraintes en utilisant les solveurs de l'outil autonome Yices. Cet outil nous a permis de vérifier la satisfiabilité de l'ensemble de contraintes qui pèsent sur le réseau TTEthernet pour générer ensuite, si c'est satisfiable, un modèle de solutions possible.

La richesse en solutions de notre adaptation de l'approche de planification nous a permis d'élaborer une étude de cas qui vise à tester l'impact de l'agencement contigu et distribué des instants d'envoi des trames TT sur l'efficacité envoi/réception et les délais de bout en bout des différents trafics véhiculant à travers le sous-système FMS. Les résultats ont montré une grande variabilité de performances : tandis que le cas d'utilisation avec un agencement contigu des instants d'envoi renvoie des résultats satisfaisant, le cas distribué des envois affiche une perte de 25% des trames critiques RC avec une augmentation drastique de son délai de bout en bout ce qui rend ce cas-ci non favorable au fonctionnement de notre système. Cette grande variabilité des performances prouve la nécessité de réviser l'ensemble

des contraintes de l'approche de planification afin de prendre en considération les nouvelles exigences de l'intégration du trafic RC.

Bien que nous ayons atteint nos objectifs en simulation TTEthernet, le processus de planification fait encore face au problème de sa dépendance envers la topologie et le plan d'échange du système cible. Il serait donc intéressant de réfléchir à une solution permettant de déduire les informations nécessaires et de les utiliser, ensuite, conjointement avec l'API Yices afin de générer d'une façon transparente le code de la planification, ce qui va mener dès lors à une automatisation complète de la simulation.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Abuteir, M., et R. Obermaisser. 2013. « Simulation environment for Time-triggered Ethernet ». In *2013 IEEE 11th International Conference on Industrial Informatics (INDIN), 29-31 July 2013*. (Piscataway, NJ, USA), p. 642-8. Coll. « 2013 11th IEEE International Conference on Industrial Informatics (INDIN) »: IEEE. < http://dx.doi.org/10.1109/INDIN.2013.6622959 http://ieeexplore.ieee.org/ielx7/6599026/6622844/06622959.pdf?tp=&arnumber=6622959&isnumber=6622844 >.

Dutertre, Bruno, Arvind Easwaran, Brendan Hall et Wilfried Steiner. 2012. « Model-based analysis of Timed-Triggered Ethernet ». In *31st Digital Avionics Systems Conference: Projecting 100 Years of Aerospace History into the Future of Avionics, DASC 2012, October 14, 2012 - October 18, 2012*. (Williamsburg, VA, United states), p. 9D21-9D211. Coll. « AIAA/IEEE Digital Avionics Systems Conference - Proceedings »: Institute of Electrical and Electronics Engineers Inc. < http://dx.doi.org/10.1109/DASC.2012.6382445 http://ieeexplore.ieee.org/ielx5/6363477/6382265/06382445.pdf?tp=&arnumber=6382445&isnumber=6382265 >.

Florin, G. *La tolérance aux pannes dans les systèmes répartis*. Laboratoire CEDRIC, 53 p.

Jean-Baptiste, Chaudron. 2013. « TTEthernet Theory and Concepts ». In *TORRENTS Project*. (Onera), sous la dir. de TTTech.

Klaus, Wehrle, Günes Mesut et Gross James. 2010. *Modeling and Tools for Network Simulation*. Verlag Berlin Heidelberg: Springer, 546 p.

Lauer, Michaël. 2012. « Une méthode globale pour la vérification d'exigences temps réel ». Institut National Polytechnique de Toulouse, 204 p.

Luxi, Zhao, Xiong Huagangh, Zheng Zhong et Li Qiao. 2014. « Improving Worst-Case Latency Analysis for Rate-Constrained Traffic in the Time-Triggered Ethernet Network ». *Communications Letters, IEEE,* vol. 18, n^{o} 11, p. 1927-1930.

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

SRI, International. 2014. *Yices 2 Manual*.

Steinbach, Till, Hermand Dieumo Kenfack, Franz Korf et Thomas C Schmidt. 2011. « An extension of the OMNeT++ INET framework for simulating real-time ethernet with high accuracy ». In *Proceedings of the 4th International ICST Conference on*

Simulation Tools and Techniques. p. 375-382. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Steiner, W. 2010. « An Evaluation of SMT-based Schedule Synthesis for Time-triggered Multi-hop Networks ». In *2010 IEEE 31st Real-Time Systems Symposium (RTSS 2010), 30 Nov.-3 Dec. 2010*. (Los Alamitos, CA, USA), p. 375-84. Coll. « Proceedings 2010 IEEE 31st Real-Time Systems Symposium (RTSS 2010) »: IEEE Computer Society. < http://dx.doi.org/10.1109/RTSS.2010.25 http://ieeexplore.ieee.org/ielx5/5701809/5702212/05702246.pdf?tp=&arnumber=5702246&isnumber=5702212 >.

Steiner, W., G. Bauer, B. Hall et M. Paulitsch. 2012. « Time-Triggered Ethernet ». In *Time-Triggered Communication*, sous la dir. de obermaisser, Roman. p. 181-220. United States of America: Taylor & Francis Group.

Steiner, W., et B. Dutertre. 2011. « Automated Formal Verification of the TTEthernet Synchronization Quality ». In *NASA Formal Methods. Third International Symposium (NFM 2011), 18-20 April 2011*. (Berlin, Germany), p. 375-90. Coll. « NASA Formal Methods. Proceedings of the Third International Symposium (NFM 2011) »: Springer. < http://dx.doi.org/10.1007/978-3-642-20398-5_27 http://download.springer.com/static/pdf/727/chp%253A10.1007%252F978-3-642-20398-5_27.pdf?auth66=1416522750_da404d9e94e4e784d0bb64768dfb47eb&ext=.pdf >.

Steiner, Wilfried, et Bruno Dutertre. 2010. « SMT-based formal verification of a TTEthernet synchronization function ». In *15th International Workshop on Formal Methods for Industrial Critical Systems, FMICS 2010, September 20, 2010 - September 21, 2010*. (Antwerp, Belgium) Vol. 6371 LNCS, p. 148-163. Coll. « Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) »: Springer Verlag. < http://dx.doi.org/10.1007/978-3-642-15898-8_10 http://download.springer.com/static/pdf/151/chp%253A10.1007%252F978-3-642-15898-8_10.pdf?auth66=1416522746_f5239b30eabc29aaf9efa3f1c16f8c5c&ext=.pdf >.

Suethanuwong, Ekarin. 2012. « Scheduling time-triggered traffic in TTEthernet systems ». In *2012 IEEE 17th International Conference on Emerging Technologies and Factory Automation, ETFA 2012, September 17, 2012 - September 21, 2012*. (Krakow, Poland), p. IEEE Industrial Electronics Society. Coll. « IEEE Symposium on Emerging Technologies and Factory Automation, ETFA »: Institute of Electrical and Electronics Engineers Inc. < http://dx.doi.org/10.1109/ETFA.2012.6489749 http://ieeexplore.ieee.org/ielx7/6479732/6489522/06489749.pdf?tp=&arnumber=6489749&isnumber=6489522 >.

Tamas-Selicean, Domitian, Paul Pop et Wilfried Steiner. 2012. « Synthesis of communication schedules for TTEthernet-based mixed-criticality systems ». In *10th ACM International Conference on Hardware/Software-Codesign and System Synthesis, CODES+ISSS 2012, Co-located with 8th Embedded Systems Week, ESWEEK 2012, October 7, 2012 - October 12, 2012*. (Tampere, Finland), p. 473-482. Coll. « CODES+ISSS'12 - Proceedings of the 10th ACM International Conference on Hardware/Software-Codesign and System Synthesis, Co-located with ESWEEK »: Association for Computing Machinery. < http://dx.doi.org/10.1145/2380445.2380518 http://delivery.acm.org/10.1145/2390000/2380518/p473-tamasselicean.pdf?ip=142.137.112.134&id=2380518&acc=ACTIVE%20SERVICE&key=FD0067F557510FFB%2E3EBA7780C0643BDE%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=601466806&CFTOKEN=86058972&__acm__=1416523039_60b59a483db42fc0f5286b42b4327747 >.

Todorov, Lazar T., Till Steinbach, Franz Korf et Thomas C. Schmidt. 2013. « Evaluating requirements of high precision time synchronisation protocols using simulation ». In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*. (Cannes, France), p. 307-313. 2512778: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Varga, Andras, et Rudolf Hornig. 2008. « An overview of the OMNeT++ simulation environment ». In *SIMUTools*. (March 03 - 07). Marseille, France.

Weingartner, E., H. vom Lehn et K. Wehrle. 2009. « A Performance Comparison of Recent Network Simulators ». In *Communications, 2009. ICC '09. IEEE International Conference on*. (14-18 June 2009), p. 1-5. < http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5198657 >.

Zhao, Luxi. 2014. « Probabilistic analysis of response latency for rate-constrained traffic in the TTEthernet network ». In *Digital Avionics Systems Conference (DASC), 2014 IEEE/AIAA 33rd*. (5-9 Oct. 2014), p. 1-26.