SDRN : réseau maillé temps réel dynamique défini par logiciel

Florian Greff

THALES UNIVERSITÉ DE LORRAINE École doctorale IAEM Lorraine

SDRN : réseau maillé temps réel dynamique défini par logiciel

THÈSE

présentée et soutenue publiquement le 16 mai 2018

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Florian Greff

Composition du jury

Président :	Dr. Yves Sorel	Directeur de recherche, Inria Paris
Rapporteurs :	Pr. Jean-Luc Scharbarg	Professeur, INPT-IRIT
	Pr. Laurent George	Professeur, ESIEE–LIGM
Examinateur :	Dr. Claire Pagetti	Ingénieur de recherche HDR, ONERA
Encadrants :	Pr. Ye-Qiong Song	Professeur, Université de Lorraine
	Dr. Laurent Ciarletta	Maître de conférences, Université de Lorraine
Invité :	M. Arnaud Samama	Ingénieur, Thales R&T (co-encadrant)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

« *Si je devais résumer ma vie aujourd'hui avec vous, je dirais que c'est d'abord des rencontres, des gens qui m'ont tendu la main. Et c'est assez curieux de se dire que les hasards, les rencontres, forgent une destinée.* »

Cette thèse n'aurait pas été possible sans ceux qui m'ont donné la chance de la réaliser. Merci à toi, Laurent, de m'avoir proposé en premier lieu cette thèse et avoir toujours cru en moi. Merci à mes encadrants, Ye-Qiong Song, Laurent Ciarletta et Arnaud Samama, de m'avoir donné la chance de réaliser cette thèse et d'avoir fait de moi un meilleur ingénieur et chercheur que je ne l'étais hier. J'ai également une pensée pour les institutions publiques (Loria, Université de Lorraine, ANRT) et entreprise (Thales Research & Technology) ayant permis et participé à la mise en place de ce projet.

Je remercie également mes collègues du laboratoire LISL de Thales Research & Technology ainsi que des équipes Madynes et Simbiot du Loria, trop nombreux pour pouvoir les citer, pour nos partages d'expériences techniques, leurs conseils professionnels et les moments amicaux passés ensemble.

Je remercie les membres de mon jury, Laurent George, Claire Pagetti, Jean-Luc Scharbarg et Yves Sorel, pour le temps qu'ils m'ont consacré lors de la lecture de ce manuscrit ainsi que lors de la soutenance, mais également pour leurs remarques, conseils et questions très pertinents, ainsi que leurs encouragements.

Enfin, je tiens à exprimer toute ma reconnaissance à mes parents, mes frère et sœur Titouan et Fanny, ma belle-famille, ma compagne Charlotte, Rourou et Chaton, pour leur soutien indéfectible avant, pendant et après cette thèse.

A mes parents,
A Charlotte,

Sommaire

Table des figures

Liste des tableaux

Publications et activités d'encadrement

Articles, conférences internationales

— Florian Greff, Ye-Qiong Song, Laurent Ciarletta et Arnaud Samama, *Combining Source and Destination-Tag Routing to Handle Fault Tolerance in Software-Defined Real-Time Mesh Networks*, RTNS'17, Grenoble, France, 2017.

— Florian Greff, Ye-Qiong Song, Laurent Ciarletta et Arnaud Samama, *A Dynamic Flow Allocation Method for the Design of a Software-Defined Real-Time Mesh Network*, WFCS'17, Trondheim, Norvège, 2017.

— Florian Greff, Eric Dujardin, Arnaud Samama, Ye-Qiong Song et Laurent Ciarletta, *A Symbiotic Approach to Designing Cross-Layer QoS in Embedded Real-Time Systems*, ERTS^2'16, Toulouse, France, 2016.

Articles, conférences nationales

— Florian Greff, Ye-Qiong Song, Arnaud Samama et Laurent Ciarletta, *Réseaux maillés dynamiques pour applications temps réel à criticités multiples : problématique et analyse*, école d'été ETR'15, Rennes, France, 2015.

Brevets

— Florian Greff, *Procédé d'optimisation d'un système embarqué et dispositifs associés*, brevet no. 18 00122, 8 février 2018.

Autres communications et posters

— Florian Greff, Ye-Qiong Song, Arnaud Samama et Laurent Ciarletta, *Software-Defined Real-Time Mesh Networking*, colloque SDN DAY, Palaiseau, France, 2017.

— Florian Greff, Ye-Qiong Song, Arnaud Samama et Laurent Ciarletta, *Software-Defined Real-Time Mesh Networking : Protocol and Experimentation Method*, école d'été RESCOM, Le Croisic, France, 2017.

— Florian Greff, Ye-Qiong Song, Laurent Ciarletta et Arnaud Samama, *Bringing Dynamicity to Real-Time Systems Through the Use of Dynamic Mesh Networking*, workshop Real-Time Networks (RTN), Bruxelles, Belgique, 2016.

— Florian Greff, Eric Dujardin, Ye-Qiong Song, Arnaud Samama et Laurent Ciarletta, *On Expressing and Guaranteeing Real-Time Quality of Service in a Dynamic Network*, école d'été ETR'15, Rennes, France, 2015.

Reviewing

— 25th International Conference on Real-Time Networks and Systems (RTNS'17), Grenoble, France, 2017

— 12th IEEE World Conference on Factory Communication Systems (WFCS'16), Aveiro, Portugal, 2016

Encadrements de stages et projets

— Stage de fin d'études d'ingénieur (TELECOM Nancy) de Romain Tissier, *Mise en œuvre des couches d'accès et de configuration d'un réseau embarqué temps réel défini par logiciel*, 2017.

— Stage technicien (ENSEM) de Vincent Martin, *Développement et intégration d'un simulateur de réseaux TSN à un framework d'expérimentation*, 2017.

— Projet d'initiation à la recherche (TELECOM Nancy), *Synchronisation du temps dans la simulation de systèmes cyber-physiques*, 2017.

— Stage technicien (TELECOM Nancy) de Romain Tissier, *Développement d'un outil graphique permettant la génération d'expérimentations de réseaux embarqués*, 2016.

Avant-propos

Ce document présente une thèse CIFRE réalisée conjointement au Loria, laboratoire d'informatique de l'Université de Lorraine, et à Thales Research & Technology. Plus particulièrement, elle a été accueillie au sein de l'équipe-projet Madynes du Loria. Cette équipe est spécialisée dans la recherche expérimentale et appliquée dédiée à la gestion de l'Internet du futur, aux réseaux de capteurs sans fil, aux systèmes cyber-physiques collaboratifs, et plus généralement aux réseaux innovants. Thales Research & Technology est le centre de R&T français du groupe Thales, structuré autour des secteurs d'activité suivants : aéronautique, espace, transport terrestre, défense et sécurité. Le groupe réunit plus de 62 000 collaborateurs dans 56 pays du monde. La thèse a été accueillie au sein du laboratoire Ingénierie Système et Logiciel du centre de recherche. Ce laboratoire est spécialisé en recherche appliquée dans les domaines de la modélisation et conception de systèmes et logiciels embarqués, et plus récemment des concepts avancés de réseaux embarqués et d'Internet des Objets. La thèse a été réalisée entre février 2015 et février 2018.

Le travail réalisé porte sur l'étude d'une nouvelle architecture embarquée à hautes performances. Dans le but d'améliorer l'utilisation du système et répondre aux besoins croissants des applications embarquées en termes de puissance de calcul et de communication, nous étudions une architecture maillée mutualisant les ressources du système. Cette architecture est également dynamique, c'est-à-dire qu'elle peut être reconfigurée pendant l'exécution. Ainsi, l'utilisation des ressources s'adapte aux besoins en temps réel. De plus, le maillage doit permettre une bonne tolérance aux défaillances par la pluralité des composants et chemins disponibles. Dans le cadre de la thèse, nous nous sommes plus particulièrement intéressés au réseau interconnectant les composants de cette l'architecture. Dans les domaines d'application étudiés (véhicules et aéronefs notamment), nous faisons face à des contraintes de sûreté, nécessitant des méthodes de garantie de certaines propriétés des communications, telles que le délai. On parle de contraintes temps réel. Les méthodes de garantie du temps réel impliquent généralement que le réseau est configuré de façon statique, de sorte à s'assurer qu'il respectera les contraintes dans tous les cas.

Scientifiquement, l'objectif est de répondre aux trois points suivants. Premièrement, une architecture supportant le changement de modes d'exécution alors que le nombre de modes croît avec l'autonomie de certains systèmes (par exemple les drones). Deuxièmement, l'allocation dynamique et incrémentale de flux de communication temps réel dans un réseau. L'allocation incrémentale permet de réduire la complexité, puisqu'elle ne considère pas l'ensemble des flux à chaque configuration. Troisièmement, l'exploitation d'une topologie maillée pour la tolérance aux fautes, sachant que l'allocation des flux temps réel doit gérer cette topologie complexe. Les approches de réseaux embarqués actuelles sont majoritairement statiques, mais des contributions scientifiques, du monde de l'embarqué ou non, proposent des concepts dynamiques de communication ou systèmes temps réel. Comme nous le détaillerons dans le premier chapitre, aucune de ces approches dynamiques ne répond à l'ensemble du problème posé : allocation incrémentale de ressources réseau, tolérance aux fautes en exploitant une topologie à multiples chemins, étude conjointe des délais pire cas et du routage.

Le **défi de la thèse** réside globalement dans le couplage entre la **configuration dynamique** du réseau et la **garantie des propriétés** des communications. A ce défi s'ajoute la gestion de la topologie maillée ou plus généralement des topologies résilientes, nécessitant un **protocole de routage** pour permettre aux données de rejoindre leur destination dans un délai imparti. Le travail a abouti à la conception d'une architecture de réseau maillé temps réel dynamique défini par logiciel, ***Software-Defined Real-time Networking***, ou **SDRN**.

De plus, le travail a été réalisé en ayant à l'esprit un objectif de transfert technologique du SDRN. Cela signifie que l'on s'assure que la solution conçue est utilisable dans un vrai système. Plus généralement, nous avons étudié l'interfaçage entre l'architecture SDRN et le reste du système embarqué.

La première contribution (Chapitre 2) est l'architecture SDRN, permettant une allocation dynamique des ressources réseau à des flux de communication, critiques ou non, dans un système embarqué [Greff et al., 2017a]. La solution est applicable à des topologies résilientes. La deuxième contribution (Chapitre 3) est un protocole de routage hybride permettant de gérer les pannes dans un système embarqué [Greff et al., 2017b]. Il répond à la problématique de tolérance aux fautes pour le SDRN, mais est également applicable à d'autres systèmes. Il améliore l'utilisation du réseau par rapport à une approche dupliquant les communications pour gérer les pannes. La troisième contribution (Chapitre 4) est une méthode de gestion des erreurs de transmission par approche probabiliste [Greff, 2018]. Elle permet d'adapter la configuration du réseau en fonction des propriétés statistiques des erreurs de transmission et d'une contrainte de fiabilité imposée par le concepteur du système. A l'instar de la précédente contribution, elle répond à la problématique de gestion des erreurs de transmission dans le SDRN, mais est également applicable à d'autres systèmes. La quatrième contribution (Chapitre 5) est une architecture d'expérimentation de réseaux [Greff et al., 2016]. Cet environnement d'expérimentation a à l'origine été conçu pour l'étude du SDRN mais s'est finalement positionné comme une contribution plus indépendante. Il permet de faire le lien entre simulateurs et plateformes d'expérimentation au sein du même environnement logiciel. Il facilite également la mise en place des expérimentations. Enfin, la cinquième contribution (Chapitre 6) est l'implémentation du SDRN sur une plateforme matérielle basée sur le protocole RapidIO. Ce travail a permis une montée en maturité du SDRN en répondant à certaines considérations industrielles et en démontrant la faisabilité de la technologie dans un système réel. Il était présent dans le cahier des charges dès l'origine du projet.

Dans un premier temps, le document présente plus en détail le contexte scientifique et l'état de l'art, puis identifie la problématique générale de la thèse. Ensuite, il présente les contributions résumées ci-dessus. Enfin, un bilan de thèse sera proposé.

Chapitre 1

Contexte, état de l'art et problématique

Sommaire

1.1 Introduction

Dans ce chapitre, nous présentons le contexte scientifique des systèmes et réseaux embarqués et temps réel dans lequel s'inscrit nos travaux. Nous présentons les défis principaux liés à ces domaines et l'état de l'art des solutions, puis identifions leurs limites, notamment en termes de dynamicité. Nous analysons ensuite l'état de l'art des **approches de configuration dynamique** de réseaux à contraintes temps réel, puis d'autres approches dynamiques plus générales utilisées en réseau. Enfin, nous décrivons le paradigme des réseaux définis par logiciels et l'état de l'art des réseaux temps réel définis par logiciel. A la lumière de ces présentations, nous identifions la problématique générale de la thèse, à savoir l'étude d'une architecture de réseau maillé temps réel dynamique défini par logiciel, SDRN.

1.2 Systèmes embarqués et problématique du temps réel

1.2.1 Introduction aux systèmes embarqués

Un système embarqué définit un ensemble de composants (notamment des capteurs, actionneurs, unités de calcul) situés à l'intérieur d'une machine, par exemple un véhicule ou un robot, et collaborant d'une façon ou d'une autre pour la faire fonctionner. Les systèmes embarqués sont généralement caractérisés par des contraintes supplémentaires par rapport aux autres systèmes informatiques. On parle notamment de contraintes SWaP, *Size, Weight and Power*, à savoir des contraintes de taille, poids et consommation d'énergie.

Les systèmes embarqués ont généralement une responsabilité plus forte en termes de sécurité humaine ou bien de missions pour lesquelles ils sont utilisés. Le cas échéant, on souhaite vérifier qu'ils fonctionnent comme prévu et fournissent des garanties de sûreté. On observe deux grandes problématiques scientifiques liées à ces aspects :

— Exactitude logique : on veut garantir que le comportement d'une application ou d'un composant sera toujours tel qu'il est souhaité par le concepteur. Notamment, si l'on considère un état du système, un environnement et des entrées identiques à des instants différents, le comportement du système doit toujours être le même. Les preuves de programmes sont l'une des nombreuses thématiques scientifiques liées à la problématique d'exactitude logique dans les systèmes embarqués ;

— Exactitude temporelle : la fiabilité du système embarqué est liée à sa capacité à garantir un temps de réponse maximale. Cela concerne à la fois le temps de réponse lié aux applications et à la façon de les exécuter, mais également aux communications entre les composants du système. Cette problématique est liée à la notion de temps réel. La thèse s'inscrit globalement dans cette problématique d'exactitude temporelle.

1.2.2 Problématique du temps réel

La notion de temps réel définit le fait qu'une donnée ou action d'un système informatique perd tout ou partie de sa valeur lorsque sa finalisation (fin de l'exécution, arrivée à destination, etc.) dépasse une échéance donnée. Par exemple, une commande de freinage n'a plus aucun sens si l'action est exécutée une minute plus tard, tandis qu'un fichier téléchargé sur Internet ne perdra pas sa valeur intrinsèque s'il arrive plus tard que prévu à destination. Bien entendu, ceci n'est valable que si l'on considère des temps de retard raisonnables, car aucun élément ne conserve sa valeur sur un temps infini.

Le temps réel n'est pas une problématique spécifique aux systèmes embarqués, on la retrouve par exemple dans la transmission d'informations multimédia, le jeu vidéo en ligne, la commande de procédés industriels, l'exécution de transactions boursières. L'ordre de grandeur des échéances varie également selon le domaine, de la microseconde pour les applications radar ou de traitement d'images dans des systèmes embarqués contraints, à l'heure pour la commande de procédés chimiques.

On distingue trois grandes familles de contraintes temps réel :

— Temps réel dur ou *hard real-time* : la tâche doit respecter son échéance sans compromis. Dans le cas contraire, l'application ou composant auquel elle est liée sera considéré comme défaillant. C'est le cas par exemple d'une commande de freinage. Cette notion est liée à la notion de criticité (cf. Section 1.2.3) ;

— Temps réel mou, souple, ou *soft real-time* : un non respect de l'échéance entraîne une perte de valeur, une perte de qualité, mais cette perte est progressive. On citera l'exemple du

jeu vidéo en ligne, où un dépassement d'une échéance est acceptable, mais l'est de moins en moins en fonction de l'importance du dépassement ;
— Temps réel ferme, ou *firm real-time* : à l'instar du temps réel dur, un dépassement de l'échéance entraîne une perte de valeur complète. En revanche, le système n'est pas considéré comme défaillant après un dépassement d'échéance. Par exemple, un système de décodage de flux vidéo est temps réel ferme, car une image n'a plus de valeur si celle qui la suit a déjà été diffusée. En revanche, la perte de quelques images est acceptable. Le ratio d'échéances pouvant être manquées dépend du système et est lié à la notion de Qualité de Service que l'on souhaite proposer à l'utilisateur.

Malgré ces différences, la notion de temps réel aboutit toujours à la recherche de la garantie de l'échéance [Xu and Parnas, 1993].

1.2.3 Criticité

La criticité d'une tâche ou d'une communication définit une forme de responsabilité qu'on lui accorde au sein du système. On distingue trois grandes catégories de criticité :
— Criticité liée à la sûreté : une défaillance entraîne la mise en danger de vies humaines ;
— Criticité liée à la mission : une défaillance entraîne la mise en danger de la mission à laquelle participe le système. Par exemple, dans un système dont le rôle est de fournir des images, le flux caméra est critique par rapport à la mission ;
— Non critique.

La criticité peut également être déclinée en plusieurs niveaux de criticité, en fonction de la qualité de la garantie de bon fonctionnement que l'on veut lui associer. Ces niveaux de criticité peuvent être liés à des contraintes de certification, notamment dans le domaine avionique.

Lorsqu'un système fait cohabiter plusieurs sous-systèmes de niveaux de cricité différents, on parle de système à criticité mixte [Burns and Davis, 2014]. Ce type de système pose des défis supplémentaires. En effet, les sous-systèmes sont de fiabilités variables car leur défaillance est plus ou moins tolérée. On souhaite donc absolument éviter que le comportement erroné de sous-systèmes non critiques ait une influence négative non prévue sur le comportement des sous-systèmes critiques.

Dans ce document, nous n'étudierons pas spécifiquement les systèmes à criticité mixte, mais proposerons une solution qui tient compte du fait que les communications peuvent avoir des criticités différentes.

1.2.4 Ordonnancement

Lorsque les ressources de calcul et de communication d'un système embarqué sont limitées et partagées par des tâches temps réel, on cherche à organiser l'utilisation des ressources de telle sorte que toutes les tâches respectent leurs contraintes temps réel. Cette problématique correspond à la théorie de l'ordonnancement [Liu and Layland, 1973] [Burns, 1991]. Il s'agit d'un domaine scientifique extrêmement vaste, fédéré par la recherche de mécanismes d'ordonnancement de tâches de contraintes différentes sur des systèmes embarqués de plus en plus complexes. Les tâches sont généralement caractérisées par leur caractéristique d'arrivée (par exemple leur période), leur échéance et le temps nécessaire pour les exécuter. Un algorithme d'ordonnancement est composé d'une méthode permettant de définir si un ensemble de tâches peut être ordonnancé [Lehoczky, 1990], ainsi que d'une méthode définissant l'ordonnancement effectif le cas échéant.

Plus récemment, la communauté scientifique s'est intéressée à l'ordonnancement de systèmes à criticité mixte. L'article fondateur [Vestal, 2007] définit l'impact du mélange des criticités

dans le mécanisme d'ordonnancement. Cet article sera suivi de [Baruah and Vestal, 2008] et [Dorin et al., 2010], conduisant à la preuve de l'optimalité de l'algorithme d'Audsley [Audsley, 1991] pour l'ordonnancement de tâches à criticité mixte sur une seule ressource, par exemple un processeur. Cet algorithme n'est cependant valable que sous l'hypothèse que les tâches peuvent être préemptées, c'est-à-dire que leur exécution peut être interrompue temporairement pour exécuter une tâche plus prioritaire, et cela sans surcoût. Cette hypothèse ne correspond pas à la réalité des systèmes, conduisant à l'analyse de méthodes d'ordonnancement sans préemption [Davis et al., 2015] ou encore la prise en compte du surcoût lié à la préemption [Yomsi and Sorel, 2007]. Une revue plus récente des méthodes d'ordonnancement de tâches à criticité mixte sur ressource unique est donnée dans [Huang et al., 2014].

Cependant, l'importance croissante de la place des processeurs multi-cœurs dans les systèmes embarqués pose de nouveaux défis d'ordonnancement des tâches sur plusieurs ressources simultanément actives [Baruah et al., 2014] [George and Hermant, 2009].

Enfin, la garantie d'un temps de réponse bout-en-bout d'une chaîne applicative distribuée, composée de plusieurs tâches et échanges entre ces tâches, nécessite l'utilisation d'approches holistiques [Tindell and Clark, 1994].

1.2.5 Tolérance aux fautes

La tolérance aux fautes regroupe les mécanismes de gestion des défaillances temporaires ou permanentes d'un composant ou d'une application du système. Elle est essentielle à la garantie du bon fonctionnement du système dans certaines conditions et doit être étudiée en parallèle de l'exactitude logique et temporelle.

On identifie deux problématiques principales dans le domaine de la tolérance aux fautes. La première est de permettre à des applications ou communications qui souffrent d'une défaillance de continuer à fonctionner. Dans ce domaine, on peut citer les solutions de redondance active ou passive (duplication), de détection et correction des erreurs, ainsi que les mécanismes de retransmission en cas d'erreur de communication. La deuxième problématique est d'isoler les défaillances de sorte qu'elles n'entraînent pas d'autres défaillances dans le système. Par exemple, si une application ne fonctionne pas comme prévu et demande à utiliser le système de façon excessive, on veut empêcher que ce dysfonctionnement déborde sur les autres. Des solutions de supervision du système et d'isolation des dysfonctionnements sont nécessaires pour répondre à cette problématique.

La tolérance aux fautes sera abordée plus en détail dans les Chapitres 3 et 4.

1.3 Réseaux embarqués temps réel

1.3.1 Problématique

Lorsqu'un système embarqué est composé de plusieurs composants distribués devant collaborer, ceux-ci communiquent entre eux par l'intermédiaire d'un sous-système de communication. Les systèmes embarqués devenant de plus en plus complexes, il est devenu nécessaire de concevoir de véritables réseaux informatiques embarqués. Ils permettent de mutualiser tout ou partie du support de communication et ainsi réduire la complexité de câblage. Des exemples typiques, que nous détaillerons plus loin, sont les réseaux avioniques avec la récente technologie AFDX et les réseaux automobiles tels que CAN. Ils proposent des méthodes déterministes d'accès au réseau, préférables dans le domaine du temps réel. Elles s'opposent aux méthodes aléatoires d'utilisation des liens, telles que CSMA/CD dans l'Ethernet non temps réel, où les interfaces gèrent les

collisions en attendant un temps aléatoire avant de retransmettre.

Les problématiques présentées précédemment s'appliquent également aux réseaux embarqués :

- **Exactitude logique** : on souhaite que le comportement du réseau soit identique dans des conditions équivalentes. On parle également de **déterminisme** ;
- **Exactitude temporelle** : certaines communications ont des contraintes temps réel, il est donc nécessaire de garantir une borne supérieure pour le temps de la communication. On parle généralement de calcul de délai pire cas ;
- **Criticité** : certains flux de communication sont critiques (commandes d'un véhicule par exemple), d'autres sont dits non-critiques ou *best-effort*. Il s'agit par exemple des flux multimédia de divertissement dans une voiture ou un avion. Lorsqu'on cherche à mutualiser le support de communication entre tous ces flux, il est nécessaire de s'assurer que les flux non-critiques n'auront pas d'influence négative non prévue sur les flux critiques ;
- **Ordonnancement** : la garantie du temps réel sur un réseau embarqué nécessite que l'on maîtrise l'organisation des différentes communications. Cela peut passer par des mécanismes d'ordonnancement sur les liens du réseau, optimisant l'accès au réseau afin que chaque flux de communication respecte ses contraintes ;
- **Tolérance aux fautes** : il est nécessaire de gérer les défaillances temporaires ou permanentes des liens, passerelles d'accès au réseau, commutateurs. On parle généralement de fiabilité ou résilience du réseau.

On pourra enfin présenter une dernière problématique clé en ce qui concerne les réseaux informatiques, il s'agit de la capacité de passage à l'échelle, ou *scalability* en anglais. L'objectif est que les mécanismes réseaux restent viables au fur et à mesure que la taille du réseau augmente. Cette thématique est fortement liée à la notion de complexité algorithmique.

Après avoir défini la notion de topologie, nous présenterons les réseaux embarqués les plus courants, les méthodes usuelles de garantie du temps réel, puis discuterons des limites des approches actuelles afin d'introduire à la problématique de la thèse.

1.3.2 Topologies

La topologie correspond à la structure géométrique du réseau. Le choix d'une topologie influe sur les possibilités de collisions entre les communications ainsi que la résilience du réseau. Cependant, les possibilités de topologies peuvent être limitées par les capacités du protocole réseau sous-jacent.

La topologie la plus simple est le bus (Figure 1.1). L'interconnexion se fait par une voie unique. Les principaux avantages de cette structure sont d'interconnecter l'ensemble des nœuds avec une faible complexité de câblage et de faciliter la distribution des informations à plusieurs éléments en même temps. Cependant, une défaillance du bus entraîne la défaillance complète du réseau. Ce point unique de défaillance oblige une redondance physique du bus entier pour la tolérance aux fautes. De plus, la centralisation du canal physique augmente le risque de collisions entre les flux. Ainsi, un système d'arbitrage central ou autre mécanisme de gestion de collisions doit être mis en place, d'où l'apparition d'un délai supplémentaire de communication, croissant avec le nombre de composants interconnectés. Pour cette raison, le passage à l'échelle est problématique.

Les réseaux commutés ont été conçus pour répondre à cette problématique de collisions et augmenter la performance des réseaux. Le trafic réseau est réparti sur plusieurs liens physiques reliant les nœuds entre eux. Chaque lien est dans un domaine de collision séparé. De la source à la destination, les paquets de données empruntent un chemin défini à l'émission ou au fur et à mesure de leur progression dans le réseau (routage). De manière générale, l'utilisation d'un réseau commuté apporte des avantages en termes de performance au prix d'une complexité protocolaire

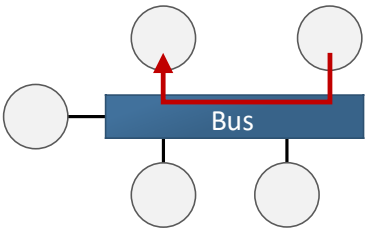

FIGURE 1.1 – Topologie bus

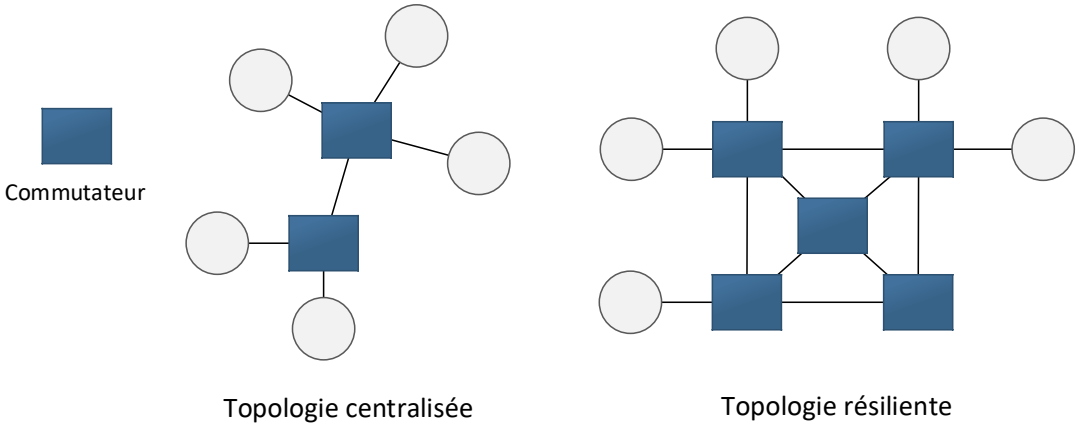

FIGURE 1.2 – Topologies commutées

et de câblage plus importante. On citera deux familles de topologies principales pour les réseaux commutés (Figure 1.2).

La première famille est la topologie centralisée. Le nombre de commutateurs est faible par rapport au nombre de composants finaux et le nombre de chemins possibles d'une source à une destination est faible également. Cette topologie élimine les collisions et peut proposer une première résilience par rapport à un bus. Elle conserve cependant une concentration des points de défaillance et nécessite une duplication des commutateurs pour la tolérance aux fautes. La topologie en étoile est un exemple de topologie centralisée.

La deuxième famille est celle des topologies résilientes, telles que les topologies maillées, toriques ou en anneau. Le nombre de commutateurs est important par rapport au nombre de composants finaux et la pluralité des chemins de communication possibles répond au problème de point unique de défaillance au niveau des éléments intermédiaires. L'augmentation de la complexité de la structure entraîne cependant une augmentation de la complexité du routage et plus généralement de l'utilisation qui est faite du réseau.

1.3.3 Réseaux embarqués usuels

PCIe et Advanced Switching

PCI Express [PCI Special Interest Group, 2014] est un standard dérivé de la norme PCI [PCI Special Interest Group, 2004] mais fonctionnant de façon commutée. Dans le monde de l'embarqué, il hérite du succès de cette dernière tout en proposant un gain de performance et de

disponibilité, ainsi qu'une détection d'erreurs plus élaborée. Son fonctionnement est asymétrique : un composant racine émet vers des composants enfants. Ceci le rend bien adapté aux communications de type carte-mère vers périphériques que l'on trouve dans les ordinateurs modernes. Cette asymétrie spécifique le rend cependant très complexe à mettre en œuvre dans de nombreux autres cas ou pour la construction de topologies spécifiques. Advanced Switching (AS) [Zirin and Bennett, 2004] est un protocole d'encapsulation se présentant comme une solution à cette limitation. Elle étend les possibilités du PCIe en termes de topologies et sens des communications [Mayhew and Krishnan, 2003].

CAN

Controller Area Network (CAN) [International Organization for Standardization, 1993] est un bus de communication temps réel initialement conçu dans le contexte de l'automobile. Son déterminisme repose sur un principe de résolution des collisions non destructif : *Carrier Sense Multiple Access/Collision Resolution* ou CSMA/CR. Le principe est le suivant. Chaque trame est préfixée par un identifiant de donnée de 11 bits dans CAN 2.0A et 29 bits dans CAN 2.0B. Lorsque plusieurs trames différentes sont émises en même temps (collision), les nœuds vont identifier la trame la plus prioritaire grâce à l'identifiant de donnée. Seule cette trame poursuit son émission. On parle d'ordonnancement à priorité fixe sans préemption. Ce mécanisme permet de résoudre les collisions sans nécessiter de contrôleur de bus. Il apporte cependant une complexité en termes d'ordonnancement lors du passage à l'échelle, puisqu'il faut définir les bons identifiants de sorte que les niveaux de priorité de chaque flux leur permettent de respecter leurs contraintes temps réel. CAN souffre également d'une limite de bande passante, puisque les trames ne peuvent pas contenir des données de plus de 127 bits (8 octets). La bande passante est alors limitée à 1 Mbit/s sur des réseaux courts (longueur inférieur à 40 mètres). L'extension CAN FD (*CAN with Flexible Data-Rate*) [Robert Bosch GmbH, 2011] a été proposée pour répondre à cette problématique de bande passante dans les réseaux automobiles. Elle permet aux trames de contenir jusqu'à 64 octets de données et au débit d'atteindre 12 Mbit/s.

Time-Triggered CAN (TTCAN) [Führer et al., 2000] est une extension imposant un ordonnancement statique contrôlé et synchronisé par un maître. Elle répond notamment à la problématique de passage à l'échelle, au prix d'un surcoût en synchronisation.

Réseaux avioniques et AFDX

Le bus ARINC 429 [ARINC, 2001] a historiquement couvert les besoins en communication dans l'avionique civile. Il s'agit d'un bus point à multipoint unidirectionnel de faible débit (moins de 100 kb/s), dupliqué pour chaque donnée à transmettre ainsi que pour la tolérance aux fautes. Face à l'augmentation de la complexité des systèmes avioniques, notamment via le concept d'Avionique Intégrée Modulaire ou IMA [ARINC, 1991], il a été nécessaire de concevoir un réseau de communication mutualisé et plus performant. L'ARINC 629 [ARINC, 1999] fut l'une des premières solutions à cette problématique, mais dont la mise en œuvre s'est avérée trop coûteuse dans le contexte avionique, au profit de l'AFDX conçu par Airbus.

L'AFDX (Avionics Full-Duplex Switched Ethernet) est un réseau Ethernet [IEEE, 2002] commuté, *full-duplex*, déterministe et redondant, basé sur le standard ARINC 664p7 [ARINC, 2009]. Il est aujourd'hui la technologie de réseau embarqué de référence dans l'avionique civile. Il est notamment utilisé à bord de l'Airbus A380. La garantie du temps réel s'appuie sur une réservation de bande passante pour les flux de communication, représentés par des liens virtuels ou VL (*Virtual Link*). Chaque VL définit un flux de communication d'une source à une destination

suivant un chemin donné. Le routage est donc totalement statique. Les VL sont caractérisés par un intervalle minimum entre l'émission de deux trames de données, le *Bandwidth Allocation Gap* (BAG), ainsi que des tailles de donnée minimale et maximale. Ces éléments servent à l'analyse temporelle du réseau. Les paquets de données sont ordonnancés selon une file d'attente (*First In First Out* ou FIFO) en sortie de commutateur.

Enfin, le standard MIL-STD-1553B [US DoD, 1978] est un bus de communication pour réseaux embarqués militaires, notamment pour l'avionique militaire. Son débit de 1 Mbit/s est faible par rapport à un réseau Ethernet. L'accès au bus est géré par un contrôleur de bus autorisant les éléments à émettre.

Une comparaison plus détaillée de ces technologies est donnée dans [Schuster and Verma, 2008].

TTEthernet

TTEthernet [SAE International, 2011] pour *Time-Triggered Ethernet* est une technologie propriétaire valorisée par la société TTTech et proposant un réseau Ethernet temps réel supportant la criticité mixte. La technologie s'appuie initialement sur l'architecture Time-Triggered [Kopetz and Bauer, 2003]. Cette architecture reprend le concept de VL décrit précédemment. Les nœuds du réseau possèdent des horloges synchronisées et chaque VL critique est autorisé à émettre dans des fenêtres temporelles précises. TTEthernet autorise également des communications contraintes en débit (*Rate Constrained* ou RC), compatibles avec les communications ARINC 664 (AFDX). Enfin, il autorise des communications de type *Best-Effort* ou BE, ce qui permet la criticité mixte.

Ethernet AVB et Time-Sensitive Networking

Ethernet Audio Video Bridging (AVB) est le standard précurseur à Time-Sensitive Networking (TSN). Son objectif était de garantir une qualité de service temps réel pour les flux de communication audio et/ou vidéo sur un réseau Ethernet. Un deuxième objectif était de permettre la coexistence de flux audio/vidéo volumineux avec des flux *best-effort*. Un troisième objectif était de proposer une réservation dynamique de qualité de service afin qu'un composant intégrant le réseau pendant l'exécution puisse utiliser ce dernier avec sûreté. Ethernet AVB est décrit dans le standard IEEE 802.1BA [TSN Task Group, 2011].

Ethernet AVB fonctionne sur le principe du *traffic shaping* (lissage du trafic). Cela consiste à allouer une partie des ressources réseau à des classes de flux et à étaler artificiellement le trafic de sorte qu'il corresponde à cette allocation. Un modèle classique du traffic shaping est le modèle dit du seau perçé (*leaky bucket*). Même si le trafic entrant arrive par rafale (dans la métaphore, des seaux d'eau versés à intervalles réguliers dans le seau perçé), le trafic sortant est lissé par l'étirement artificiel du flux. En résumé, les classes de flux temps réel ont la priorité sur les flux *best-effort*, mais sont limitées dans le trafic. Cela permet de garantir une partie des ressources réseau aux flux *best-effort*, empêchant ainsi leur famine. Le *traffic shaping* utilisé dans Ethernet AVB est décrit dans le standard IEEE 802.1Qav [TSN Task Group, 2016b]. Le problème de synchronisation temporelle des nœuds du réseau est quant à lui adressé par le standard IEEE 802.1AS [TSN Task Group, 2016a] et le protocole Precision Time Protocol [IEEE, 2008].

Le groupe de travail sur Ethernet AVB a été renommé en Time-Sensitive Networking (TSN) afin de poursuive les recherches mais surtout d'aborder quatre points majeurs :

— Le passage d'un contrôle décentralisé vers un contrôle centralisé ;
— La correction d'un manque de fiabilité dans le standard 802.1Qav ;
— La préemption des trames Ethernet pour optimiser l'utilisation du réseau ;

— L'extension des domaines d'application en dehors de l'audio/vidéo.

Nous pouvons également noter l'ajout d'une nouvelle classe de trafic, *Control Data Traffic* (CDT). TSN devient aujourd'hui la technologie de référence pour la mise en œuvre de réseaux Ethernet temps réel embarqués, notamment dans le domaine de l'automobile [Lim et al., 2012]. Elle s'appuie sur le *Time Aware Traffic Shaping* décrit dans le standard IEEE 802.1Qbv [TSN Task Group, 2016c]. En résumé, la vie du réseau est organisée selon des cycles qui se répètent, eux-mêmes divisés en fenêtres temporelles. Chaque fenêtre temporelle autorise le passage de paquets répondant à un critère (par exemple, un niveau de priorité) et selon un ordonnancement défini. Il est possible d'appliquer un ordonnancement Ethernet AVB sur un réseau TSN, les deux technologies sont donc parfaitement compatibles. TSN ayant vocation à être utilisé dans un ensemble hétérogène de domaines, il fait l'objet de nombreuses recherches, notamment sur l'ordonnancement des flux de communication [Specht and Samii, 2017]. Le routage est également un point scientifique saillant, puisque la configuration des chemins de communication nécessite de coupler les méthodes d'analyse temporelle spécifiques à l'ordonnancement utilisé et les méthodes de recherche de chemins [Nayak et al., 2017]. La tolérance aux fautes est généralement apportée par la duplication des flux critiques sur des chemins disjoints [Gavrilut et al., 2017].

Réseaux sur puce

Les réseaux sur puce, *Networks-on-Chip* ou NoCs, sont un cas particulier des réseaux embarqués. Il s'agit de la technologie interconnectant les éléments d'un système sur puce, par exemple les cœurs du processeur à la mémoire, ou les nombreux cœurs d'un processeur many-core entre eux [Bjerregaard and Mahadevan, 2006]. Dans ce type de processeurs notamment, le nombre croissant de cœurs a nécessité la conception de réseaux sur puce à topologie complexe telles que des topologies maillées. Ces nouvelles topologies apportent de bonnes propriétés de résilience et mutualisation des ressources réseau, mais également des défis en termes de routage et gestion de la congestion. En particulier, le routage dans les NoCs fait l'objet de nombreuses recherches [Wu et al., 2016], mais nous aborderons ce point plus en détail dans le Chapitre 3.

Tous les NoCs ne sont pas temps réel, nous nous concentrons ici sur ceux qui le sont. [Hesham et al., 2017] propose une revue récente et complète des technologies de NoCs temps réel et l'utilisation de topologies résilientes dans ce domaine. En couplant cette revue et celle de [Mello et al., 2009], on identifie cinq techniques principales d'utilisation du réseau dans le cas des NoCs temps réel :

— **Surdimensionnement** : le réseau est surdimensionné, ce qui trivialise l'analyse du temps réel. Cette technique perd de son intérêt avec l'augmentation des besoins réseau des nœuds et de la compétitivité industrielle. Elle reste néanmoins un moyen trivial de résoudre le problème du temps réel dans les réseaux embarqués ;
— **Commutation de circuits** : lorsque deux nœuds du NoC souhaitent communiquer, un chemin complet est réservé de bout-en-bout. Il n'y a donc aucun risque d'interférence, au prix d'un possible surdimensionnement du chemin réservé ;
— **Multiplexage temporel** : le principe est de répondre au problème de surdimensionnement des circuits présenté ci-dessus. Plutôt que de réserver les liens complets sur le circuit, on va les partager entre des circuits virtuels. Ces derniers utilisent les liens chacun à leur tour. Ce partage peut être effectué de façon statique [Lu and Jantsch, 2007] ou dynamique [Concer et al., 2011] ;
— **Commutation de paquets** : les données sont divisées en paquets comme dans un réseau IP, ce qui permet une utilisation plus fine du réseau. En revanche, contrairement à un réseau IP, les tampons d'un NoCs sont très petits, ils ne permettent même pas le sto-

ckage d'un paquet complet. Pour cette raison, une méthode appelée *wormhole switching* a été imaginée. Les paquets sont divisés en sous-paquets appelés *flits* qui se suivent et peuvent s'étaler sur plusieurs tampons [Dally and Towles, 2003]. Cette méthode de commutation optimise l'utilisation du réseau et répond au problème de sous-dimensionnement des tampons, mais peut créer de nombreuses situations de blocages. Un paquet peut en effet bloquer plusieurs tampons s'il subit une congestion à un endroit du réseau. Pour cette raison, elle est très complexe à mettre en œuvre [Shi and Burns, 2008]. L'utilisation de canaux virtuels est une solution à ce problème. Le principe est de diviser les liens en canaux virtuels, de sorte qu'un paquet n'en bloque qu'un seul, ce qui facilite le contournement de la congestion pour les autres et réduit ainsi grandement les interférences [Dally and Towles, 2003]. On citera également les recherches sur les topologies des NoCs, par exemple la topologie *flattened butterfly* facilitant le contournement des congestions [Kim et al., 2007].

— **Techniques hybrides** utilisant deux ou plusieurs des techniques précédentes.

[Tobuschat et al., 2013] donne un aperçu des techniques pouvant être mises en œuvre dans un NoC pour répondre aux problématiques de tolérance aux fautes et de criticité mixte. Le principe clé est d'isoler les tâches et flux de communication par le principe de la virtualisation afin de confiner les dysfonctionnements.

RapidIO

RapidIO [RapidIO Trade Association, 2014] est un standard pensé pour l'embarqué et utilisé dans la plupart des stations 3G/4G, l'imagerie médicale et le calcul hautes performances. Ses objectifs sont un débit très élevé (plusieurs dizaines de Gbit/s), des mécanismes de réduction de la latence et une fiabilisation des communications par un système d'acquittement point-à-point ou *store-and-forward*. Les symboles de contrôle (*Control Symbols* ou CS), utilisés pour le contrôle de flux, peuvent être insérés dans le flux de données, ce qui permet une réactivité maximale et donc une réduction de la latence globale. Par exemple, l'information de corruption d'un paquet peut être transmise sans attendre la fin de la transmission du paquet en cours.

RapidIO offre également la possibilité d'assigner des priorités et canaux virtuels aux flux de communication. En revanche, les politiques d'utilisation de ces informations doivent être définies par le concepteur du système. Le routage est également à la discrétion du concepteur, mais le protocole propose des champs d'adresses source et destination dont les tailles peuvent atteindre 32 bit, soit plus de 2 milliard d'adresses. Un contrôle de flux bout-en-bout peut être effectué selon divers moyens (basé sur des crédits, XON/XOFF, etc.)

On citera également [Fuller, 2005], un livre décrivant de façon complète le standard RapidIO.

SpaceWire/SpaceFibre

SpaceWire [Parkes, 2012] et sa version optique, SpaceFibre, sont des technologies dédiées au spatial, utilisées ou envisagées dans de grandes missions spatiales : ExoMars, Swift, Rosetta, Lunar Orbiter, etc. Cette orientation se traduit notamment par l'utilisation du *wormhole switching* présenté plus haut. Ce mécanisme est également lié à la taille des tampons, contrainte par l'environnement difficile spécifique au domaine spatial. Le routage est effectué soit de façon classique, par la combinaison de l'adresse du destinataire et de tables de routage, soit par la spécification du chemin complet au niveau de l'émetteur. La philosophie générale du procole est proche de celle de RapidIO, notamment en ce qui concerne la fiabilisation des communications ainsi que le contrôle de flux point-à-point. Néanmoins, certaines fonctionnalités ne sont pas spécifiées :

configuration des commutateurs, gestion des priorités, *multicast*, notification de paquets perdus au niveau de la couche applicative de l'émetteur. Enfin, SpaceWire contient un protocole de synchronisation temporelle.

1.3.4 Méthodes d'évaluation et de garantie du temps réel

Une problématique centrale des réseaux embarqués est la garantie du respect des contraintes temps réel des flux de communication. Dans ce but, il est nécessaire d'être capable d'analyser les délais bout-en-bout pire cas, de sorte que le temps réel soit garanti en toutes circonstances. On parle d'analyse temporelle ou *timing analysis* en anglais. En plus du délai pire cas, le domaine de l'analyse temporelle s'intéresse également aux garanties de bande passante et de gigue [1].

L'objectif de cette partie n'est pas d'analyser en profondeur la problématique d'analyse temporelle mais de décrire les approches principales utilisées dans ce domaine et comprendre leurs limites.

Premièrement, on peut classer les approches selon qu'elles sont locales ou globales. Les approches locales n'analysent pas l'ensemble du système d'un coup mais composent le résultat global à partir d'analyses de sous-parties, par exemple le délai pire cas subi pour la traversée d'un nœud. Les approches globales ou holistiques analysent le système dans son ensemble et appliquent généralement la politique du *pay burst only once*, c'est-à-dire qu'on fait la différence entre les délais subis à chaque saut et un délai de rafale subi une seule fois de bout-en-bout. En réalité, de nombreuses approches s'appuient à la fois sur des méthodes d'analyse locale et des méthodes d'analyse globale.

Les approches d'analyse temporelle se divisent en trois groupes principaux :

— Les approches calculant un délai pire cas exact par vérification de modèle ;
— Les approches calculant une borne supérieure du délai pire cas. Ce problème est connu comme étant NP-difficile [Bouillard et al., 2010], d'où l'utilisation d'approximations qui augmentent le pessimisme. Le défi principal de ces approches est de réduire ledit pessimisme [Boyer and Fraboul, 2008], éventuellement par des approches probabilistes [Scharbarg et al., 2009]. Les trois approches principales de ce groupe sont le calcul réseau, la méthode des trajectoires et l'analyse de performances par composition ;
— Les approches calculant une distribution du délai bout-en-bout, typiquement par simulation ou en utilisant la théorie des files d'attentes. De cette distribution peuvent être déduites les valeurs de délais pire cas, ou des valeurs optimistes associées à une probabilité de dépassement.

Vérification de modèle

La vérification de modèle, ou *model checking* [Berard et al., 2010], est une méthode d'analyse temporelle s'appuyant sur un modèle temporisé du système, généralement un automate temporisé [Alur and Dill, 1994]. Elle est utilisée dans [Charara et al., 2006] pour le calcul de délais pire cas exacts dans un contexte de réseau avionique AFDX. Elle est cependant susceptible à l'explosion combinatoire et passe donc difficilement à l'échelle.

Calcul réseau

Le calcul réseau, ou *Network Calculus*, à l'origine présenté par [Cruz, 1991], compte également comme textes fondateurs [Le Boudec and Thiran, 2001], [Chang, 2000] et [Jiang and Liu, 2008].

1. La gigue définit la variation du délai. On parle de *jitter* en anglais.

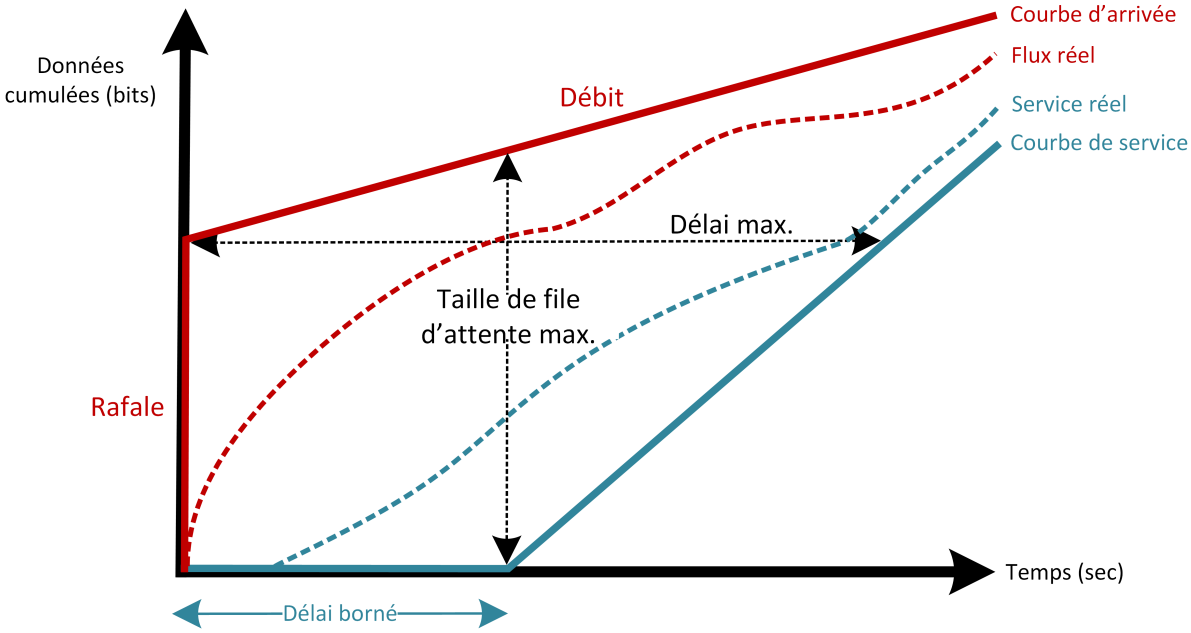

FIGURE 1.3 – Courbes d'arrivée et de service

Le système que l'on souhaite analyser est divisé en sous-systèmes, autrement appelés nœuds. Le niveau de granularité est variable. On peut également considérer un seul nœud pour l'ensemble du système, ce qui revient à appliquer une approche globale ou holistique.

Les flux en entrée des nœuds sont représentés par des courbes d'arrivée, bornes supérieures des comportements réels des flux, plus complexes à représenter et analyser. Les capacités de traitement en sortie sont représentées par des courbes de service minimum. En composant ces courbes localement puis globalement, on est capable de calculer des valeurs de délai et taille des files d'attente pire cas pour les flux considérés (Figure 1.3).

Méthode des trajectoires

La méthode des trajectoires est une approche d'analyse temporelle non holistique. Elle étudie en effet le délai pire cas subi par un flux ou paquet le long de sa trajectoire dans le réseau et ne considère pas le reste du système. Elle est moins pessimiste qu'une approche holistique pour les systèmes distribués [Martin et al., 2005]. Elle a été appliquée à l'analyse de temps de réponse pire cas dans le cas d'ordonnancements FIFO [Martin and Minet, 2006] et FP/EDF (Fixed Priority/Earliest Deadline First) [Martin et al., 2004].

Les travaux de [Bauer et al., 2010] sur l'analyse temporelle des réseaux AFDX ont montré que la méthode des trajectoires était moins pessimiste que le calcul réseau, mais que la combinaison des deux pouvait encore améliorer le résultat de l'analyse.

Le reproche principal qui a été fait à cette méthode est son possible optimisme, c'est-à-dire que la borne de délai calculée est, dans certains cas, inférieure au pire cas réel. Ces problèmes ont été identifiés par [Kemayo et al., 2013] et des solutions sont proposées par [Li et al., 2014].

Analyse de performances par composition

L'analyse de performances par composition, ou *Compositional Performance Analysis* (CPA), est une méthode d'analyse des pire et meilleurs cas d'exécution dans les systèmes embarqués [Henia et al., 2005]. Comme son nom l'indique, il s'agit d'une approche locale obtenant les résultats bout-en-bout par composition des résultats obtenus au niveau des sous-systèmes. Elle opère une abstraction de la granularité précise des événements entrant et sortant des sous-systèmes, les agrégeant sous la forme de flux caractérisés par des valeurs de périodes et gigues. Ce mécanisme lui confère un très bon passage à l'échelle. En revanche, l'abstraction est source de pessimisme, car l'analyse prend en compte des scénarios impossibles. Par exemple, elle peut considérer le cas où une tâche attend sur un processeur l'exécution d'une tâche qui la précède dans la chaîne d'exécution, ce qui est impossible car la tâche considérée ne peut exister que si la précédente a déjà été exécutée.

L'approche CPA peut être utilisée spécifiquement pour l'analyse de délais réseau pire cas avec le modèle approprié, comme dans [Thiele and Ernst, 2016] par exemple. Cependant, le problème évoqué plus haut est exacerbé dans le cas d'un réseau à commutation de paquets à topologie complexe. En effet, la transmission bout-en-bout d'un paquet est modélisée sous la forme d'une longue chaîne de tâches, chacune correspondant au passage d'un nœud ou lien par le paquet. Lors d'un ordonnancement FIFO par exemple, si deux paquets arrivent en même temps, l'un va attendre la transmission de l'autre puis passera derrière. Il ne paiera donc pas une deuxième fois cette attente due à l'arrivée simultanée. Or, CPA considèrera que, dans le pire cas, les paquets arrivent en même temps à chaque saut, ce qui exacerbe grandement le pessimisme dans ce type de réseaux.

Simulation

Cette approche consiste à simuler tous les scénarios possibles ou un échantillon représentatif, dans le but d'obtenir une distribution des délais (Figure 1.4). A partir de cette distribution, une borne supérieure peut être déduite. Cette approche offre une bonne compréhension des résultats puisqu'ils sont associés aux scénarios détaillés. En revanche, cette recherche de l'exhaustivité la rend très difficile à passer à l'échelle.

Les approches par simulation ont notamment été mis en œuvre par [Dolejs and Hanzalek, 2003], [Scharbarg and Fraboul, 2007] et [Mifdaoui et al., 2007] pour l'analyse de réseaux Ethernet temps réel.

1.4 Premières limites de réseaux embarqués : introduction à la problématique de la thèse

Nous identifions quelques premières limites des approches de réseaux embarqués actuelles, sur lesquelles s'appuiera la construction de la problématique de la thèse.

1. Allocation dynamique de ressources aux flux temps réel

Les systèmes embarqués sont généralement configurés de façon statique. Il est donc nécessaire d'allouer aux applications des ressources de calcul et de communication telles qu'elles pourront s'exécuter efficacement au moment où elles en demanderont le plus. Or, ces ressources ne sont pas toujours utiles en même temps. Le cas échéant, le système et plus particulièrement le réseau sont surdimensionnés par rapport à une configuration où les ressources sont mutualisées et

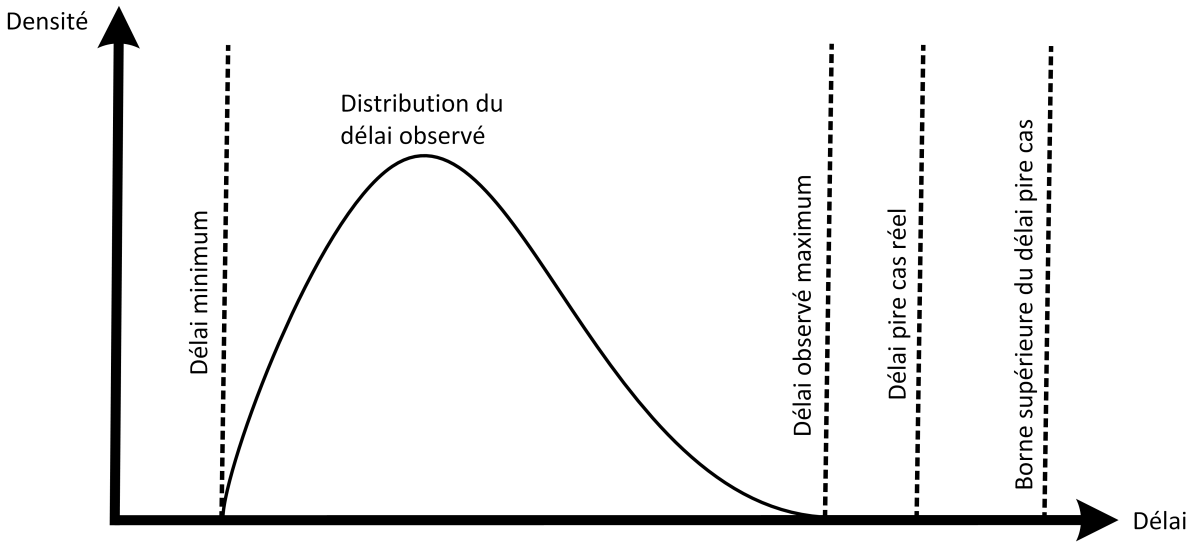

FIGURE 1.4 – Distribution des délais par simulation

allouées en fonction de la situation. Ce problème de surdimensionnement, associé au besoin en synchronisation temporelle pour certains réseaux, rend compliqué le passage à l'échelle.

2. Optimisation de l'utilisation du réseau pour diminuer le surdimensionnement

L'objectif de garantie du temps réel dans le pire cas peut introduire un surdimensionnement du système. Ceci a plusieurs raisons possibles : surdimensionnement pour trivialiser l'analyse du pire cas, valeur de pire cas excessive par rapport à la moyenne mais possible, utilisation de bornes supérieures au vrai pire cas. Les approches *(m,k)-firm* [Ramanathan, 1999] et *weakly-hard real-time* [Bernat et al., 2001] sont une réponse au problème du pire cas difficile mais rare. Elles considèrent que les applications peuvent manquer une échéance de temps en temps, ce qui permet de réduire le pessimisme de la configuration. Une autre solution est de fournir une garantie probabiliste [Scharbarg et al., 2009].

3. Tolérance aux fautes

Les topologies embarquées, à l'exception des réseaux sur puce, sont généralement centralisées. Lorsqu'on veut garantir la tolérance aux fautes, les éléments réseau sont dupliqués. La majorité du temps, les éléments de secours ne sont pas utilisés. Dans cette thèse, nous explorons l'apport d'une topologie résiliente telle qu'une topologie maillée aux systèmes embarqués. Dans ce type de topologie, les liens et nœuds utilisés en mode nominal peuvent également servir d'éléments de secours. On obtient ainsi une mutualisation de la redondance physique.

4. Prise en compte de la fragmentation

Les méthodes d'analyse temporelles citées ne prennent pas directement en compte la fragmentation des données. On prend l'hypothèse d'une taille maximale de trame ou de paquet dont le temps réel est garanti. Or, des applications embarquées modernes telles que des applications radar ou utilisant des capteurs plus puissants s'échangent des données dont la taille peut dépasser cette taille maximale. Dans ce cas, on souhaite que l'analyse temporelle prennent en compte la fragmentation nécessaire et le surcoût lié, de sorte que la couche applicative soit déchargée de cette responsabilité. Une première approche naïve consiste à multiplier le délai pire cas d'un paquet par le nombre de paquets nécessaires pour envoyer la donnée. La Figure 1.5 donne un

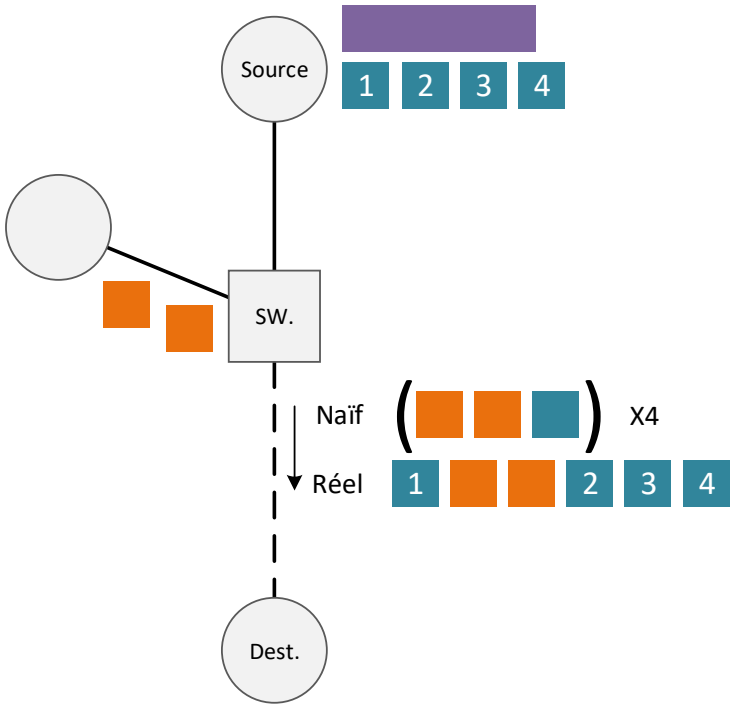

FIGURE 1.5 – Illustration de la prise en compte de la fragmentation

exemple simple d'une donnée de quatre paquets traversant un commutateur. Avec l'approche naïve, on estime que, dans le pire cas, un paquet bleu doit attendre la transmission des deux paquets oranges avant la sienne. Le délai pire cas de transmission d'un paquet bleu est donc de 3 unités. On en déduit un délai pire cas de transmission de la donnée, soit 12 unités. En réalité, les paquets d'autres flux n'entrent pas forcément en conflit avec chaque paquet de la donnée. De plus, une fois arrivé sur le nœud suivant, le premier paquet n'est pas obligé d'attendre le dernier avant de poursuivre sa transmission. L'ensemble de ces considérations font que l'approche naïve est pessimiste par rapport à la réalité. De plus, ce pessimisme est exacerbé à chaque saut et devient trop important sur une topologie complexe. Pour répondre à ce problème, les deux inconnues principales que l'on doit calculer sont, d'une part, la quantité de données retardant l'envoi du premier paquet de la donnée, d'autre part, la quantité de données du même flux ou non devant être transmise entre le premier et le dernier paquet. L'évaluation de ces quantités est un problème que nous traiterons dans le Chapitre 2.

Ces considérations nous amènent à étudier une plateforme embarquée dynamique à topologie résiliente. L'objectif est d'être capable d'allouer, pendant l'exécution, les ressources de calcul et réseau aux applications, en sachant que celles-ci ont des contraintes et criticités diverses. On parlera d'allocation **en ligne** des ressources. Plus particulièrement, nous nous intéressons au réseau d'une telle architecture. La problématique de cette thèse est donc la conception d'une architecture de communication embarquée dynamiquement configurable et capable de garantir le respect des contraintes temps réel. Le réseau doit être une plateforme de communication sûre et résiliente, au service de la couche applicative. Pour des raisons de passage à l'échelle, on souhaite que cette architecture puisse fonctionner de façon asynchrone, sans besoin en synchronisation temporelle. De plus, s'agissant d'une étude amont qui n'est pas liée à des cas d'utilisation spécifiques, l'architecture ne doit pas imposer le protocole sous-jacent utilisé. Plus précisément, on

souhaite identifier les mécanismes protocolaires nécessaires à la mise en place d'un tel réseau temps réel dynamique. La généricité de l'approche par rapport au protocole sous-jacent nécessite une prise en compte de la fragmentation, dont l'évaluation du délai pire cas de l'ensemble de la donnée applicative, et non plus d'un fragment (paquet), est un nouveau défi en soit.

Dans les cadres d'applications possibles, on pourra également citer la conception de véhicules multi-missions, par exemple un drone pouvant effectuer des missions différentes. La capacité de reconfigurer le système en ligne permet au véhicule d'enchaîner plusieurs missions différentes. Ceci est possible en pré-configurant plusieurs modes de configuration statiques, mais la configuration dynamique permet un passage à l'échelle en termes de nombres de modes et d'adaptation aux environnements.

Nous allons maintenant revoir les approches de configuration dynamique dans les systèmes embarqués, en se concentrant sur les réseaux embarqués. Puis nous décriront d'autres approches de configuration et réservation dynamique de ressources réseau. En ayant à l'esprit les contraintes propres à l'embarqué et au temps réel ainsi que les limites de ces approches dynamiques, nous identifierons plus précisément la problématique à résoudre pour la conception d'une architecture de communication temps réel, dynamique et agnostique des couches basses du système.

1.5 Approches dynamiques dans les systèmes embarqués

Des recherches s'intéressent à l'apport du dynamisme aux architectures embarquées, que ce soit pour répondre à des problématiques d'optimisation des performances ou de tolérance aux fautes. De manière générale, le couplage entre le dynamisme et la flexibilité d'une part, et les contraintes de sûreté et temps réel d'autre part, est un problème difficile.

Les approches dynamiques sont parfois associées à la notion de contrôle d'admission. Il s'agit d'un processus de validation vérifiant la faisabilité d'une allocation de ressource avant d'autoriser la communication ou l'exécution d'une application. Il se base sur des paramètres ou contraintes indiquées en entrée de processus et doit généralement prendre en compte l'état du système.

[Abdelzaher et al., 2000] décrit un *framework* de négociation des ressources de calcul allouées aux applications d'un système embarqué. Il se base sur la notion de dégradation contrôlée. Les applications peuvent avoir plusieurs niveaux de Qualité de Service, chacun associé à des points de récompense si le niveau est alloué, et des points de pénalité si aucun niveau n'est alloué. Ce système permet d'optimiser le déploiement des applications et d'adapter celui-ci en ligne. Il permet également à des applications de tourner en mode dégradé, alors qu'elles seraient complètement éteintes dans un système classique.

[Neukirchner et al., 2010] présente une approche de négociation orientée contrats. Elle propose une interface de programmation homogène permettant aux applications d'exprimer leurs contraintes. Si le contrôle d'admission est positif, les contraintes sont matérialisées sous la forme d'un contrat, que le *framework* de gestion et supervision du système fait respecter. L'article aborde la notion de latence de reconfiguration et l'influence significative de la couche réseau sur le temps de reconfiguration. Un article suivant, [Neukirchner et al., 2011], se concentre sur l'utilisation de l'approche dans un contexte de criticité mixte. L'objectif est de garantir le respect des contrats des tâches les plus critiques, alors que certaines tâches peuvent dysfonctionner et utiliser plus de ressources que demandé. La virtualisation et le lissage du trafic sont deux réponses majeures à cette problématique. Elles permettent une isolation des tâches et des communications, protégeant ainsi les autres. Des mécanismes supplémentaires sont chargés de superviser et contenir les dysfonctionnements.

La dynamicité des systèmes embarqués apporte également des problématiques de spécification

des mécanismes de reconfiguration [Krichen et al., 2010] et d'implémentation de la reconfiguration dans les applications [Rasche and Poize, 2005].

1.5.1 Approches de réseaux embarqués dynamiques

Flexible Time-Triggered (FTT) [Pedreiras and Almeida, 2003] est un paradigme de communication basé sur des cycles temporels synchrones (*time-triggered*). Son avantage est de permettre la reconfiguration dynamique via un ordonnancement dynamique effectué par un nœud maître. Chaque cycle, ce nœud maître envoie un message de déclenchement (*Trigger Message*) à tous les nœuds esclaves pour synchroniser leurs émissions. FTT peut être appliqué sur plusieurs protocoles réseau tels que CAN et Ethernet. Une architecture plus haut niveau a été proposée pour permettre l'utilisation des mécanismes FTT sur n'importe quel type de plateforme [Noguero et al., 2012]. Nous pouvons noter trois limitations principales à ce paradigme. Premièrement, une latence de communication accrue, à cause de l'envoi des messages de déclenchement avant toute communication. Deuxièmement, FTT suppose le routage statique, ce qui empêche la valorisation de topologies résilientes ou la migration d'applications. Troisièmement, la tolérance aux fautes est basée sur la duplication active des composants et communications [Derasevic et al., 2013]. Là encore, ce principe ne valorise pas la redondance passive apportée par les topologies résilientes.

Dans [Nayak et al., 2016], les auteurs proposent un réseau TSN défini par logiciel, Time-Sensitive Software-Defined Networking (TSSDN). Il s'agit d'un réseau dynamique capable de fournir des garanties temps réel pour les communications de type *time-triggered* dont l'ordonnancement peut être reconfiguré en ligne. Il aborde également le routage des flux temps réel. En effet, l'article discute du fait que la plupart des approches font l'hypothèse que les routes sont connues *a priori*. Ceci peut résulter en une sous-utilisation du réseau, notamment dans le cas où de nombreux flux sont routés à travers un goulot d'étranglement au lieu d'être distribués sur des chemins différents. L'évaluation du réseau inclut l'évaluation du temps de reconfiguration, dans ce cas le temps de calcul des tables d'ordonnancement. Cette contrainte temporelle sur la reconfiguration est l'une des problématiques clé des systèmes embarqués dynamiques. Le protocole souffre de problèmes de passage à l'échelle pour deux raisons. La première est l'utilisation de communications synchrones. La deuxième est l'incapacité à allouer de façon incrémentale des flux de communication sans remettre en cause les flux déjà alloués. Le fait de devoir prendre en compte l'ensemble du système à chaque contrôle d'admission est de manière générale un facteur limitant des systèmes embarqués dynamiques. Pour cette raison, nous proposerons une approche asynchrone et incrémentale.

[Tobuschat et al., 2013] présente un réseau sur puce à criticité mixte. L'article aborde la notion de communication agnostique des tuiles physiques. Cela signifie qu'une application ne sait ni sur quelle tuile physique elle est située, ni avec quelle tuile physique elle communique. Ce principe de virtualisation du réseau permet de reconfigurer le réseau de façon transparente du point de vue des applications. Ces mécanismes de communication agnostique assouplissent la configuration du réseau et facilitent le passage à l'échelle des méthodes de reconfiguration.

[Jouy et al., 2014] aborde l'allocation dynamique de bande passante pour des flux non critiques au sein d'un réseau AFDX. Le routage utilise les composants dupliqués du réseau afin d'obtenir la meilleure utilisation possible. L'approche est cependant limitée en regard de notre objectif, car elle ne fonctionne que sur réseau AFDX et pour des flux non critiques. En effet, le délai est pris en compte comme un critère de discrimination entre plusieurs chemins et non comme une contrainte stricte. [Zhou et al., 2011] propose un contrôle d'admission de liens virtuels AFDX basé sur le calcul réseau. Il n'aborde pas la problématique de routage.

TSN propose, au sein du standard 802.1Qcc [TSN Task Group, 2016f], des améliorations au protocole SRP (Stream Reservation Protocol) [IEEE 802.1 AVB TG, 2010] originaire d'Ethernet AVB. Ce protocole permet de réserver les ressources nécessaires à un flux de communication le long de son chemin. Il met à contribution les nœuds intermédiaires. Il est utile pour allouer de façon sûre quelques flux de communication temps réel. Cependant, l'absence de contrôle global ne permet pas une optimisation de l'utilisation du réseau. Les améliorations apportées dans le cadre de TSN concernent la fiabilité et la performance du protocole de réservation de ressources. Elles permettent également aux acteurs du réseau de s'échanger *a priori* des informations sur leurs présences et compatibilités. Enfin, le standard IEEE 802.1Qca [TSN Task Group, 2016d] propose des fonctionnalités permettant d'optimiser les chemins empruntés par les flux (par exemple, au lieu de prendre systématiquement le plus court, on pourrait répartir la charge) et de les adapter en cas de panne d'un composant intermédiaire.

1.6 Qualité de Service dans les réseaux IP

Certaines applications utilisant l'Internet ou les réseaux IP sont sensibles à la Qualité de Service et profitent des garanties temps réel. On peut citer par exemple le jeu vidéo en ligne et les flux audio et vidéo. Pour répondre à ces demandes de qualité de service, des solutions de contrôle d'admission et réservation de ressources ont été proposées. [Firoiu et al., 2002] donne une revue complète des travaux sur la Qualité de Service dans l'Internet. Les deux architectures principales sont Integrated Services (IntServ) et Differentiated Services (DiffServ).

IntServ [Clark et al., 1994] a une approche déterministe orientée flux. Chaque flux est traité indépendamment et possède sa propre queue dans les nœuds du réseau. Cette granularité permet de garantir le respect de contraintes temps réel dur aux flux mais pose des problèmes de passage à l'échelle dans des réseaux très grands tels que l'Internet. Le protocole le plus utilisé pour la demande et réservation effective des ressources est RSVP [Zhang et al., 1997]. Il nécessite que les éléments intermédiaires soient compatibles.

DiffServ [Davies et al., 1998] a une approche orientée classes de trafic. Il ne prend pas en compte les flux individuellement mais les agrège dans des classes de trafic. Ceci facilite le passage à l'échelle mais complexifie la garantie de la Qualité de Service pour des flux précis. Le passage à l'échelle est encore amélioré par le fait que les paquets sont classifiés par les routeurs en bordure de domaine. Les routeurs du cœur de réseau ne prennent pas de décision et ne traitent que les classes de trafic (pas de queue pour chaque flux). Des recherches ont étudié comment utiliser ce paradigme orienté classes pour garantir une Qualité de Service temps réel à des applications. [Charny and Boudec, 2000] discute l'impact de l'ordonnancement aggrégé sur le délai pire cas. [Schmitt et al., 2003] propose une méthode de garantie du temps réel en utilisant les valeurs de priorités comme classes de trafic et une approche calcul réseau pour le calcul du délai pire cas. Les auteurs expliquent que le passage à l'échelle est très limité par l'explosion des combinaisons d'interactions possibles à chaque nœud. Au final, le gain en capacité de passage à l'échelle apporté par DiffServ est grandement réduit lorsque l'on cherche à garantir du temps réel dur. Cette conclusion conforte l'intérêt d'une allocation incrémentale des flux de communication dans le but de linéariser la complexité.

1.7 Réseaux définis par logiciel (SDN)

Le réseau défini par logiciel, ou *Software-Defined Networking*, ou SDN, est un paradigme récent consistant à centraliser la configuration du réseau au niveau d'un serveur ou logiciel

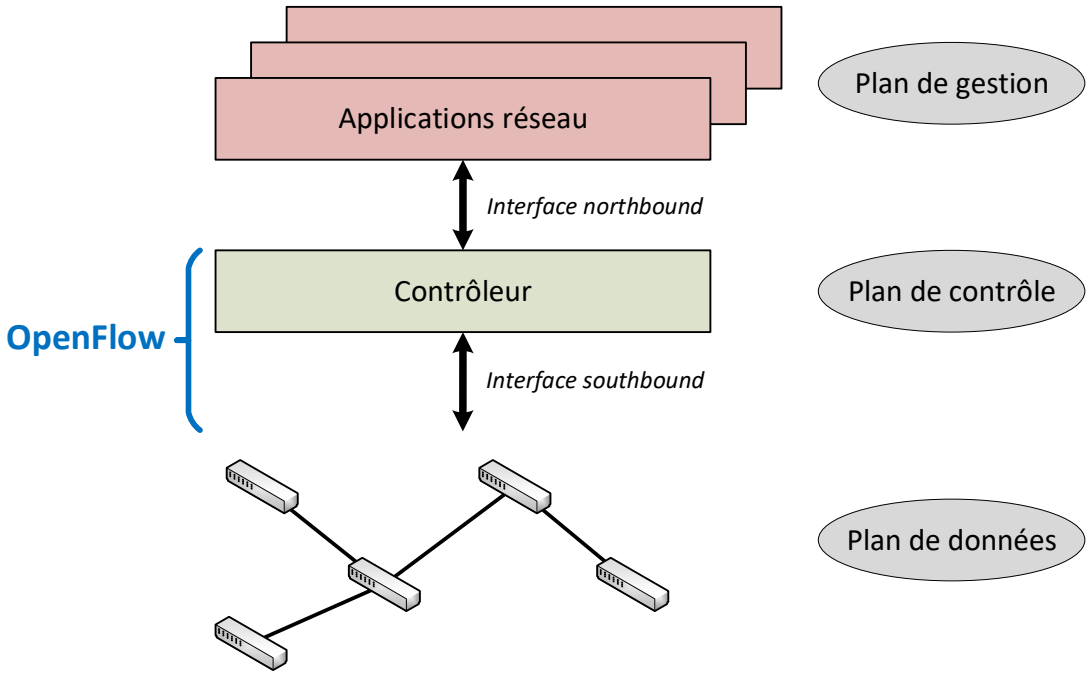

FIGURE 1.6 – Architecture SDN

externe. Plutôt que d'embarquer l'intelligence de contrôle du réseau dans les équipements, on l'externalise et la centralise. Plus précisément, le SDN sépare le plan de données, constitué des équipements physiques du réseau, du plan de contrôle et de configuration.

On distingue typiquement trois plans au sein d'un SDN (Figure 1.6) :

— **Le plan de données** : il s'agit du réseau physique, des équipements qui vont manipuler les paquets ;

— **Le plan de contrôle** : un ou plusieurs contrôleurs capables de configurer le réseau physique par l'intermédiaire d'une interface dite *southbound*. L'interface la plus connue et utilisée est le protocole OpenFlow [McKeown et al., 2008], standardisé par l'Open Networking Foundation ;

— **Le plan de gestion** ou applications réseau : ont une vision éventuellement simplifiée du réseau et prennent des décisions haut niveau sur la configuration. Elles recoivent la vue du réseau et dirigent le contrôleur par l'intermédiaire d'une interface dite *northbound*. Le contrôleur est ensuite chargé de faire la configuration effective. Il n'existe pas, à l'heure actuelle, d'interface *northbound* standard. Ce point fait l'objet de nombreuses recherches. [Humernbrum et al., 2014] en propose par exemple une dédiée aux applications interactives temps réel.

[Kreutz et al., 2015] est une revue très complète des réseaux définis par logiciel. Il existe notamment de nombreuses recherches sur l'apport de Qualité de Service dans les réseaux SDN et OpenFlow, via IntServ, DiffServ, ou d'autres approches (par exemple [Sonkoly et al., 2012]). Nous nous concentrons dans la suite de cette partie sur l'utilisation de SDN pour les systèmes critiques et le temps réel dur.

[Lei Du and Herlich, 2016] discute des intérêts de SDN pour les réseaux Ethernet temps réel. La globalisation du contrôle permet d'optimiser l'utilisation du réseau et donc d'allouer plus de

flux de communication et/ou des flux nécessitant plus de bande passante. Elle facilite également le travail de routage grâce à la connaissance haut niveau de l'état du réseau. Ceci permet de valoriser les topologies complexes et donc les topologies résilientes telles que maillées. Enfin, le fait qu'un seul élément doive gérer la reconfiguration dynamique facilite son passage à l'échelle. L'article cite deux limites principales à l'utilisation de SDN dans les réseaux temps réel dur. La première est le manque de mécanismes dans OpenFlow pour gérer le temps réel à bas niveau, car il n'a initialement pas été conçu pour cela. La deuxième est le fait que la centralisation du contrôle crée un point unique de défaillance. Ce défaut peut être géré par une redondance du contrôleur, mais cela peut apporter des défis supplémentaires.

[Ternon et al., 2016] propose d'utiliser OpenFlow pour orchestrer les commutateurs dans un réseau Flexible Time-Triggered (FTT). Les décisions du maître FTT sont matérialisées par le contrôleur SDN. Cette approche permet d'améliorer certains points de FTT, notamment la gestion des flux sporadiques.

Comme nous l'avons abordé plus haut, le problème principal de l'utilisation de SDN pour les réseaux embarqués est le manque de mécanismes permettant de garantir le temps réel : lissage du trafic, ordonnancement, isolation, etc.

Le version 1.3 d'OpenFlow propose un premier mécanisme limité de lissage de trafic. Il est possible de spécifier une bande passante maximale pour un flux en bits ou paquets par intervalle de temps. Si un paquet du flux fait dépasser le ratio, il est mis en attente ou jeté (selon la configuration). [Heise et al., 2015] propose une utilisation de ce mécanisme pour reproduire le comportement sûr de l'AFDX sur des composants pris sur étagère. Le calcul réseau utilisé pour l'AFDX est adapté à ce lissage de trafic. L'étude parvient à reproduire les liens virtuels AFDX sur un commutateur OpenFlow mais ne s'est pas montrée très concluante en termes de performances.

[Silva et al., 2017] est un travail de recherche en cours proposant une extension OpenFlow pour pouvoir spécifier des services temps réel, notamment la notion de *time-triggered* pour une utilisation dans FTT. Il est lié aux travaux de [Ternon et al., 2016] cités plus haut.

La théorie des files d'attente [Kleinrock, 1977] est largement utilisée dans l'analyse de performances des réseaux IP définis par logiciel. Elle s'intéresse au cas moyen, à la distribution statistique de la performance et la stabilité des files d'attentes. [Azodolmolky et al., 2013] explique que cette approche n'est pas très adaptée à l'analyse du pire cas dans les SDN temps réel. Il montre comment le calcul réseau peut être utilisé pour déduire des bornes supérieures des délais pire cas dans un réseau SDN, en complément de la théorie des files d'attentes. De manière générale, la plupart des approches SDN raisonnent en termes de bande passante ou de délai mou mais pas de délai dur, nécessaire dans les systèmes embarqués.

Lorsque l'on souhaite configurer dynamiquement un réseau temps réel à topologie résiliente, il est donc nécessaire de traiter en même temps la problématique d'analyse temporelle et la problématique de routage des flux de communication. Nous décrivons maintenant des approches SDN répondant à cette problématique.

[Guck and Kellerer, 2014] utilise la programmation linéaire en nombres entiers pour optimiser l'allocation des flux de communication sur le réseau tout en garantissant le respect de leurs contraintes temps réel. La limite de l'algorithme présenté est sa complexité, le rendant inutilisable en ligne.

[Kumar et al., 2017] rappelle que l'algorithme de routage de flux temps réel ne doit pas chercher le plus court chemin dans le réseau, mais un chemin qui permette aux données d'arriver à destination à temps. La première contribution de l'article est de justifier l'isolation des flux dans des queues différentes pour que les délais bout-en-bout soient stables malgré la présence

et l'apparition d'autres types de trafic dans le système. C'est une notion que nous réutiliserons dans nos travaux. La deuxième contribution est le développement de mécanismes de garantie du temps réel à des flux alloués sur un réseau SDN. Les algorithmes présentés se placent au niveau du plan de gestion et communiquent avec un contrôleur OpenFlow par une interface *northbound*. Malheureusement, OpenFlow ne permet pas d'installer les queues des flux, il est donc nécessaire de le faire directement sur les équipements de façon *ad hoc*. Ceci limite l'intérêt de SDN en termes de centralisation de la configuration. Le contrôle d'admission revient à une recherche de chemin sous plusieurs contraintes, ici les contraintes de délai et de bande passante. C'est également une notion que nous réutiliserons.

[Xu et al., 2015] présente une approche similaire, synthétisée sous la forme d'un problème de recherche de chemin sous contraintes. Le délai est modélisé de façon simple car l'article ne traite pas de l'analyse temporelle. Il justifie néanmoins l'intérêt de discrétiser les valeurs de délai pour simplifier la recherche de chemin, ce que nous ferons dans nos travaux.

Enfin, RT-SDN [Oh et al., 2016] combine un ordonnancement à priorités fixes avec une analyse temporelle holistique. L'approche passe difficilement à l'échelle à cause de la méthode d'ordonnancement. En effet, la distribution des priorités nécessite de réétudier l'ensemble des flux à chaque reconfiguration. L'admission d'un flux seul sur un trafic existant n'est étudiée que pour un flux possédant uniquement une contrainte de bande-passante. Cette hypothèse simplifie grandement l'analyse, car la priorité la plus faible peut être assignée au flux, ce qui supprime l'interférence potentielle avec les autres. Ces considérations renouvellent l'intérêt pour une méthode d'allocation incrémentale et agnostique du trafic futur.

1.8 Conclusion

Dans ce chapitre, nous avons présenté le contexte des systèmes embarqués et des problématiques de temps réel. Nous nous sommes ensuite concentrés sur les réseaux embarqués et les méthodes de garantie du temps réel. Cette revue nous a permis d'identifier l'intérêt pour une plateforme embarquée, à topologie résiliente, et capable de se reconfigurer de façon dynamique. Cette plateforme répond notamment aux problèmes de surdimensionnement des systèmes embarqués (cf. Section 1.4). Pour des raisons de simplicité de langage, on parlera généralement de réseau maillé temps réel dynamique. Nous avons présenté les approches de réseaux temps réel dynamiques. Enfin, nous avons décrit le paradigme des réseaux définis par logiciel (SDN) et les recherches s'intéressant à cette approche pour les systèmes embarqués. Cette approche est séduisante, car nous souhaitons que notre réseau maillé dynamique soit configuré en fonction de besoins et d'une stratégie proches de la couche applicative. La plateforme que nous imaginons est alors complétée par cette notion. Elle devient un réseau maillé temps réel dynamique défini par logiciel. En anglais, cela devient ***Software-Defined Real-time mesh Networking*, SDRN**.

Certaines des approches que nous avons présentées répondent partiellement à la problématique que nous nous sommes fixée. Nous avons cependant identifié un ensemble de points clé auxquels aucune approche ne répond complètement.

1. Passage à l'échelle et utilisation en ligne

Le réseau doit passer à l'échelle en termes de nombres de nœuds et de flux de communication. De plus, la configuration dynamique doit être suffisamment rapide pour être utilisable en ligne. Pour ces raisons, on souhaite que l'influence de l'état du réseau sur la complexité du contrôle d'admission soit négligeable. Cela suppose que l'on est capable d'opérer une **allocation incrémentale** des flux de communication, c'est-à-dire que l'allocation d'un flux ne remette pas en cause le reste de la configuration. Ce point est l'un des points de différenciation avec les ap-

proches SDN temps réel décrites dans la section précédente. Pour améliorer encore le passage à l'échelle, nous envisageons une **approche asynchrone**.

2. Réseau défini par logiciel intégrant le temps réel dur

Comme nous l'avons vu, le paradigme SDN a été initialement conçu pour les réseaux à applications non temps réel ou temps réel mou. Il en est de même pour la plupart des approches à Qualité de Service garantie, généralement centrées sur la garantie de bande passante. Pour les systèmes que nous visons, nous devons garantir à la fois la bande passante et le délai bout-en-bout de façon stricte. Certaines des approches que nous avons présentées traitent de ce sujet.

3. Routage et analyse temporelle conjoints

La topologie maillée nécessite de traiter conjointement les problématiques de routage et d'analyse temporelle, généralement étudiées séparément. La littérature nous apprend qu'il s'agit d'un problème complexe.

4. Prise en compte de la fragmentation

Des applications modernes comme les applications de traitement radar communiquent des données volumineuses ne pouvant être contenues dans un seul paquet ou trame. Elles doivent donc être fragmentées en plusieurs unités. Les approches d'analyse temporelles étudient généralement le délai pire cas sur un paquet, laissant la responsabilité de la gestion de la fragmentation ou imposant une taille de donnée maximale aux couches supérieures. Nous souhaitons que notre réseau maillé soit capable de garantir à une application que ses données arriveront à temps, quelque soit la taille de ces dernières (à supposer que les ressources réseau le permettent). Cela nécessite d'analyser la façon dont s'opère la fragmentation en fonction du protocole sous-jacent et de la prendre en compte dans l'analyse temporelle. Nous nous intéressons au temps d'arrivée de l'ensemble de la donnée utile.

5. Approche agnostique du protocole sous-jacent

Le transfert d'une technologie vers le monde industriel peut être complexe, surtout dans le domaine des systèmes à contraintes de sûreté. Certains protocoles, tels que CAN dans l'automobile, sont bien établis. Imposer un protocole spécifique, fusse-t-il très standard comme Ethernet, risque de limiter le transfert technologique. Cela limite également l'évolution du réseau bas niveau vers des protocoles plus performants qui pourraient faire leur apparition à l'avenir. Pour ces raisons, nous envisageons une approche générique et modulaire. Elle doit être capable de s'adapter au protocole sous-jacent. Elle doit également permettre le remplacement de certaines parties, par exemple l'ordonnancement, par d'autres implémentations, ce sans remettre en cause le reste de l'architecture. Plus généralement, on cherche à identifier les mécanismes nécessaires à bas niveau pour réaliser un réseau maillé temps réel dynamique défini par logiciel.

6. Implémentation sur plateforme RapidIO

Un objectif est de prouver la faisabilité industrielle du SDRN par son implémentation sur une plateforme maillée basée sur le protocole RapidIO (cf. Section 1.3.3). Le choix du protocole est une contrainte industrielle de Thales Research & Technology et ne sera pas justifié précisément dans le document. De manière générale, il a été choisi pour son très haut débit, la présence de champs protocolaires ouverts dans les entêtes, et l'existence d'implémentations *open source*. Ce protocole ne concerne que l'implémentation et la simulation et ne sera pas limitant dans l'étude de l'architecture SDRN. Il s'agit seulement d'une base de réseau réel pour l'expérimentation. Nous aurons néanmoins à l'esprit cette plateforme RapidIO lorsque nous aborderons les aspects expérimentaux.

En synthèse, ces points justifient l'étude d'une architecture modulaire d'allocation incrémen-

tale de flux temps réel sur un réseau à topologie résiliente, prenant en compte le délai, la bande passante et la fragmentation. Cette architecture, SDRN, doit être implémentée de bout-en-bout sur une plateforme maillée RapidIO pour prouver sa faisabilité.

La suite du document sera organisée comme suit. Le Chapitre 2 décrit l'architecture SDRN sous hypothèse de fiabilité. Le Chapitre 3 présente une approche de routage originale pour la gestion des pannes. Le Chapitre 4 présente une approche probabiliste pour la gestion des erreurs de transmission. Le Chapitre 5 décrit ERICA, un *framework* d'expérimentation hybride mêlant aspects réels et simulés et conçu initialement pour l'expérimentation sur SDRN. Enfin, le Chapitre 6 présente quelques points saillants de l'implémentation.

Chapitre 2

SDRN : fonctionnement général et allocation de flux

Sommaire

2.1 Introduction

Ce chapitre présente l'architecture *Software-Defined Real-time Networking*, ou SDRN. Il s'agit d'une méthode générique de configuration dynamique d'un réseau maillé temps réel, en fonction de besoins exprimés par la couche applicative. Son principe général est de permettre à des applications d'envoyer des requêtes à une entité centralisée, le contrôleur, présent sur un nœud du réseau. Ces requêtes contiennent les caractéristiques et contraintes correspondant à des flux

de communication que la couche applicative souhaite mettre en place. Le contrôleur est chargé d'analyser ces requêtes de façon dynamique puis fournir une réponse aux applications. Si la couche applicative confirme l'allocation des flux de communication, le réseau est configuré en cohérence avec l'analyse effectuée par le contrôleur. Sinon, une phase de dialogue supplémentaire est entamée avec le contrôleur (non traité dans la thèse). Globalement, on conçoit le réseau comme une entité au service de la couche applicative, et dont le contrôleur est l'intermédiaire. Ce dernier doit être capable de fournir une analyse très rapide de la configuration réseau à adopter, afin que le processus d'initialisation des communications soit viable à l'exécution. De plus, la complexité d'utilisation du réseau doit être transparente du point de vue de la couche applicative. En termes d'interactions, la couche applicative va dialoguer avec le contrôleur dans un premier temps, puis, une fois le feu vert donné par celui-ci, communiquer directement sur le réseau, avec l'assurance que ses contraintes seront garanties. Notre approche va ainsi plus loin que les mécanismes de Qualité de Service présents dans certains intergiciels comme Data Distribution Service [Object Management Group, 2014], qui ne permettent que de spécifier les contraintes temps réel et notifier la couche application du respect ou non ces contraintes, sans mécanisme de garantie à bas niveau. Une fois les communications établies, le contrôleur SDRN continue cependant de superviser le réseau en tâche de fond, afin de le reconfigurer et notifier la couche applicative en cas de dysfonctionnement. Enfin, nous prévoyons que les requêtes transitent par un canal de communication dédié, de sorte à ne pas interférer avec le trafic existant.

Une originalité de notre étude est de s'intéresser à la mise en réseau maillé des composants, dans le but de valoriser la multiplicité des chemins de communication obtenue par une telle topologie. Cela implique de traiter les problématiques de routage de façon dynamique.

Ainsi, l'architecture SDRN est une solution bout-en-bout d'utilisation dynamique d'un réseau temps réel à topologie résiliente (par exemple maillée). En son centre, le contrôleur est un chef d'orchestre utilisant ses connaissances du réseau bas niveau et des besoins applicatifs pour définir une stratégie de configuration la plus optimisée possible. Dans notre contexte de configuration dynamique du système, on définit une configuration optimisée comme une configuration garantissant le respect de leurs contraintes aux flux de communication considérés, tout en étant la plus altruiste possible envers les autres objectifs du système : par exemple, flux de communication futurs, répartition de charge, degré de tolérance aux fautes, etc. Par son aspect modulaire, l'architecture SDRN est capable de s'adapter aux propriétés du protocole sous-jacent et du matériel, sans imposer de contrainte supplémentaire aux applications. Un élément de l'architecture peut être remplacé par un autre, à condition qu'il respecte certaines propriétés. De plus, la méthode d'analyse temporelle permet d'allouer les flux de façon incrémentale. Cela signifie que l'on n'analyse pas l'ensemble du système à chaque nouvelle allocation de flux, ce qui apporte une grande souplesse au système. Enfin, le fonctionnement général est asynchrone, il ne nécessite donc aucune synchronisation temporelle entre les composants du réseau.

Une représentation basique de l'architecture SDRN est donnée en Figure 2.1. Dans ce chapitre, nous détaillons sa construction et son utilisation. En revanche, ce chapitre ne traite pas des problématiques liées à l'implémentation en contexte réel (interface applicative et utilisation du matériel), qui feront l'objet du Chapitre 6.

Après avoir défini notre modèle d'étude, nous détaillerons le module d'ordonnancement du SDRN, puis le module contrôleur. Ensuite, nous proposerons une application numérique. Enfin, nous mènerons une réflexion par rapport aux réseaux définis par logiciel en général, puis une synthèse de la contribution et de ses perspectives.

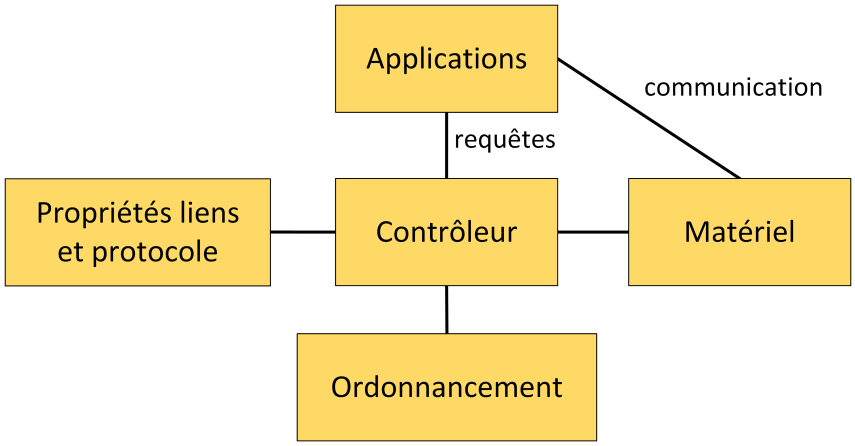

FIGURE 2.1 – Représentation basique de l'architecture SDRN

2.2 Modèle d'étude

2.2.1 Réseau, nœuds et liens

Le réseau $\mathcal{R}$ est représenté sous la forme d'un graphe orienté. Ses sommets sont les nœuds et ses arcs sont les liens reliant les nœuds entre eux. Le modèle du réseau, $\mathcal{G}_{\mathcal{R}}$, contient également les informations sur son état : flux et applications présentes, ressources déjà allouées, état de charge des liens, etc.

Chaque nœud est un commutateur réseau, relié ou non à un processeur ou tuile de calcul (CPU, GPU, DSP, etc.) par un port 0 et un bus interne. On le note N et identifié par un nombre unique dans le réseau, compris entre 1 et m, m étant la taille du réseau. On a donc $\mathcal{R} = \{N_1, \ldots, N_m\}$. Un nœud est composé d'un ou plusieurs ports physiques externes par lequel il est relié à d'autres nœuds. Les ports non exploités sont ignorés dans le modèle. La Figure 2.2 illustre le modèle d'un nœud possédant quatre ports externes et une tuile processeur.

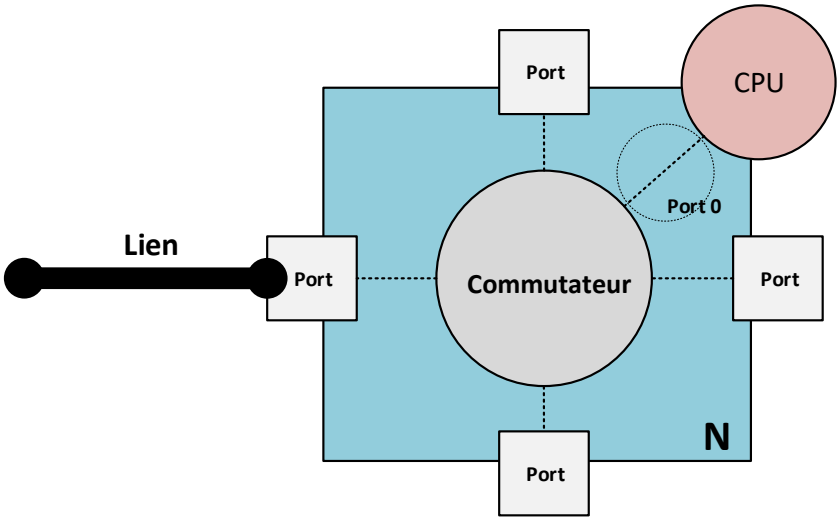

FIGURE 2.2 – Exemple de nœud : modèle

Chaque lien L est orienté, on représentera donc les liens *full-duplex* sous la forme d'un couple

de liens orientés dans des sens opposés. On définit :

— $L_{i,s\to t}$ le lien interne reliant le commutateur vers la tuile de calcul sur le nœud N_i ;
— $L_{i,t\to s}$ le lien interne reliant la tuile de calcul vers le commutateur sur le nœud N_i ;
— $L_{i\to j}$ le lien externe reliant le nœud N_i vers le nœud N_j.

Ces notions sont illustrées en Figure 2.3.

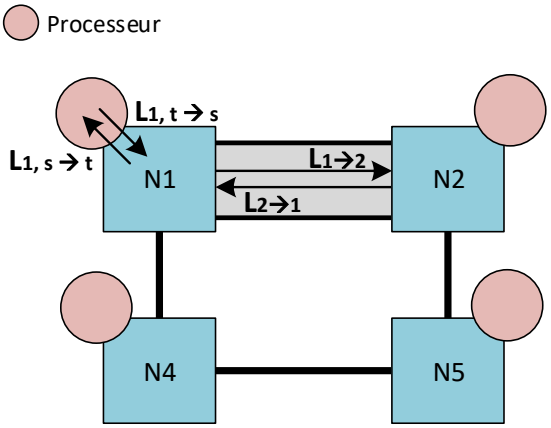

FIGURE 2.3 – Illustration du modèle de liens

2.2.2 Flux et requêtes

Un flux de communication correspond à une communication régulière ou non entre deux applications, de variables ou de messages concernant un sujet donné. Un exemple simple est la communication, d'une application A vers une application B, de la température mesurée par un capteur. Si l'application A communique également une valeur de vitesse à une application B, de manière non synchronisée avec les données de température, alors cela correspond à un deuxième flux. Cette notion est proche des liens virtuels définis par le standard ARINC-664p7 [ARINC, 2009] et utilisés dans les réseaux Avionics Full-Duplex (AFDX) et Time-Triggered Ethernet [SAE International, 2011].

Les données envoyées par les applications sont éventuellement fragmentées puis encapsulées par des entêtes réseau pour former des paquets réseau. On appelle **donnée** la charge utile brute envoyée par l'application avant fragmentation et encapsulation.

Le principe du SDRN est d'allouer des ressources réseau à des flux de communication f possédant les propriétés suivantes : $propriétés_f = \{type_f, T_f, S_f, dl\_req_f, src_f, dest_f\}$, où :

— $type_f$ est le type de flux : dans un premier temps, nous considérons des flux soit temps réel périodiques (PRT pour *Periodic Real-Time*) soit non temps réel (NRT pour *Non Real-Time*). La périodicité est un modèle du pire cas, les flux sporadiques peuvent donc être modélisés avec ces caractéristiques ;
— T_f est la période minimale, en secondes. On peut l'assimiler à un *Bandwidth Allocation Gap* dans AFDX (cf. Chapitre 1) ;
— S_f est la taille de donnée maximale, en bits. On rappelle qu'il s'agit des données brutes sortant de la couche applicative. Il ne s'agit pas d'une contrainte liée au réseau mais d'une taille que les données du flux ne dépasseront pas ;
— dl_req_f est l'échéance relative (*deadline requirement* en anglais) entre la transmission du premier bit et l'arrivée du dernier bit de la donnée, en secondes ;

— src_f est le nœud source. Il s'agit du nœud sur lequel est situé le processeur exécutant l'application émettrice ;
— $dest_f$ est le nœud destination. Il s'agit du nœud sur lequel est situé le processeur exécutant l'application réceptrice.

Les caractéristiques de période et taille sont optionnelles pour les flux non temps réel. La caractéristique d'échéance n'est évidemment pas utilisée pour ces flux.

Les requêtes de flux envoyées au contrôleur par les applications contiennent les propriétés du flux : $requête_f = \{type_f, T_f, S_f, dl\_req_f, src_f, dest_f\} = \{propriétés_f\}$.

Enfin, l'ensemble des flux communicant sur le réseau est noté $\mathcal{F}_{\mathcal{R}}$.

2.2.3 Propriétés matérielles

Le débit utile maximal d'un lien L est noté C_L, en bit/s.

2.3 Hypothèses

Nous faisons quelques hypothèses à des fins de simplification.

Premièrement, les transmissions ne subissent aucune erreur ni perte et les composants sont fiables. La gestion des pannes sera étudiée en Chapitre 3 et la prise en compte des erreurs de transmission fera l'objet du Chapitre 4.

Deuxièmement, les paquets sont commutés plus rapidement que le total des débits des liens entrants (on parle de *full line speed switches*). Cela signifie qu'il ne peut y avoir d'attente en entrée de commutateur, en revanche il peut y avoir une attente en sortie de nœud quand plusieurs paquets de ports différents sont commutés vers le même port de sortie. Cette caractéristique est en pratique partagée par la plupart des commutateurs Ethernet. La latence de commutation d'un paquet entre le port d'entrée et le port de sortie est représentée par une constante $d_{sw,\mathcal{R}}$.

Troisièmement, les liens externes sont identiques entre eux, et les liens internes sont identiques entre eux, mais éventuellement différents des liens externes. On parlera de réseau homogène. On note alors un lien externe quelconque LE et un lien interne quelconque LI. Lorsque les liens externes ou internes ont des caractéristiques différentes, on parlera de réseau hétérogène.

Quatrièmement, les communications ne s'effectuent qu'en *unicast*. Cela signifie qu'un flux de communication a une destination unique. Dans le cas où une application doit transmettre la même donnée à plusieurs applications destinataires (*multicast*), cela sera représenté sous la forme de plusieurs flux possédant les mêmes caractéristiques sauf la destination. Cette méthode peut aboutir à une utilisation inutile de ressources réseau sur les parties communes des chemins, avant que les flux ne se séparent pour rejoindre leurs destinations respectives. En effet, les paquets seront doublés sur les liens concernés au lieu de ne l'être qu'à l'embranchement. Cependant, la prise en compte et l'optimisation du *multicast* sont, de manière générale, des problèmes très complexes. Nous décrirons donc l'architecture SDRN sur la base d'une modélisation des flux en *unicast* seulement, puis apporterons une réflexion concernant l'extension pour le *multicast*.

Enfin, nous faisons l'hypothèse que les demandes d'allocation de flux sont suffisamment sporadiques et légères pour qu'elles ne constituent pas un goulot d'étranglement. Par exemple, il ne peut y avoir plusieurs centaines de requêtes en même temps. Le réseau peut supporter tous les envois de requêtes et le contrôleur peut le traiter sans attente significative.

2.4 Principes généraux du SDRN

SDRN est conçu comme une architecture modulaire mettant le contrôleur en son centre. Celui-ci est chargé d'analyser les requêtes de flux en tenant compte des propriétés du réseau et des propriétés du mécanisme d'ordonnancement des paquets à bas niveau. Les informations échangées entre les modules sont appelées les **interfaces**. Un module (dit module abstrait) a plusieurs implémentations possibles, aussi appelées modules concrets. Un module concret peut être remplacé par un autre à condition qu'il respecte les interfaces du module abstrait.

L'architecture SDRN (Figure 2.4) est constituée des modules abstraits suivants :

— Applications : représente la couche applicative présente sur le réseau ;
— Contrôleur : analyse les requêtes de flux et les propriétés du réseau. Fournit des réponses aux applications et des informations de configuration au matériel ;
— Ordonnancement : mécanisme bas niveau d'isolation des flux en sortie de nœud ;
— Propriétés liens et protocole : propriétés des liens internes et externes, ainsi que du protocole sous-jacent utilisé pour communiquer sur ces liens ;
— Matériel : architecture matérielle de la plateforme sous-jacente.

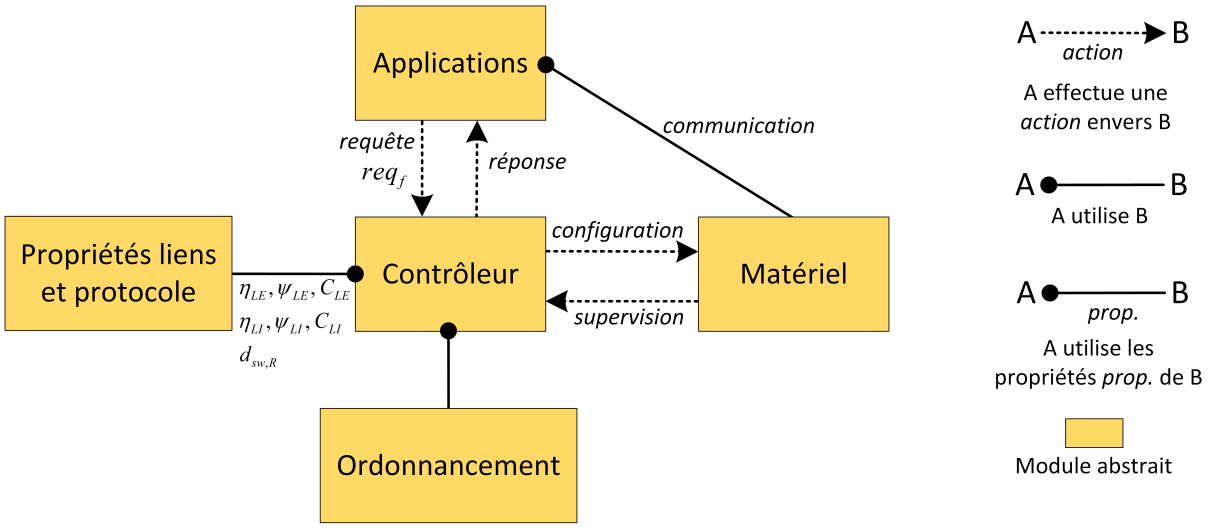

FIGURE 2.4 – Architecture SDRN non détaillée

Dans la suite de ce chapitre, nous détaillons cette architecture, les modules abstraits, interfaces, et proposons une solution SDRN concrète bout-en-bout.

2.5 Ordonnancement

2.5.1 Module abstrait

Le système de gestion des flux à bas niveau doit permettre leur allocation et désallocation dynamique sur le réseau, sans interférer avec les autres flux. L'allocation incrémentale étant l'une des forces de l'approche SDRN, il est en effet très important que les applications ne soient pas affectées par les périodes transitoires (allocation de flux, gestion de panne, etc.) des autres.

Le module abstrait d'ordonnancement représente ce mécanisme de transmission maîtrisée des paquets en sortie de nœud. Il doit proposer une isolation stricte des flux, c'est-à-dire qu'une utilisation minimale du réseau doit pouvoir être garantie pour un flux donné. Cela signifie que

l'on doit être capable d'analyser le comportement pire cas du flux sans hypothèse sur les flux précédemment alloués et ceux qui seront possiblement alloués plus tard.

Plus précisément, les propriétés nécessaires au module d'ordonnancement sont :

— *Isolation* : depuis le point de vue d'un flux, le comportement théorique des autres flux reste valable même si ces derniers ont un comportement erroné en réalité. Les comportements erronés ne peuvent interférer plus que ce que le modèle théorique a prévu. Cette propriété rend l'allocation incrémentale possible ;

— *Déterminisme* : le résultat de l'ordonnancement sera toujours le même pour des entrées identiques ;

— *Équité minimale* : l'ordonnancement est suffisamment équitable entre les flux pour que les valeurs de son interface avec le contrôleur soient calculables et finies en toute circonstance. En d'autres termes, la famine est impossible.

Paramètres et fonction protocolaires

Le module d'ordonnancement peut avoir besoin d'utiliser des paramètres particuliers pour fonctionner ; par exemple, un niveau de priorité pour chaque paquet. L'ensemble des paramètres d'un flux nécessaires au bon fonctionnement du module d'ordonnancement est appelé **paramètres protocolaires** et est noté $\rho_{f,\mathcal{R}}$. Le module d'ordonnancement doit fournir la fonction permettant de calculer les paramètres protocolaires minimum requis d'un flux f dans le réseau R, telles que les contraintes de bande passante du flux seront respectées en toute circonstance (quelques soient les allocations suivantes sur le réseau) :

$$\rho(f, \mathcal{R}) = \rho : f, \mathcal{R} \rightarrow \rho_{f,\mathcal{R}} \tag{2.1}$$

La fonction $\rho(f, \mathcal{R})$, appelée **fonction protocolaire**, fait partie de l'interface entre le module d'ordonnancement et le contrôleur (Figure 2.7). Il peut également s'agir d'un algorithme.

Fragmentation

Un avantage important de l'architecture SDRN est sa capacité d'adaptation aux caractéristiques du protocole sous-jacent. En effet, la fragmentation des données est analysée et aucune contrainte de taille maximale des données n'est imposée.

Sur un lien L, une donnée du flux f, de taille maximale S_f, est fragmentée en paquets dits de tête, de taille identique $\mu_{f,L}$, suivis éventuellement par un paquet de queue de taille $\epsilon_{f,L} < \mu_{f,L}$. Le protocole permettant la communication sur un lien L impose une taille de donnée maximale au sein de chaque paquet : la charge utile maximale, notée ψ_L. Ce fragment de donnée est ensuite encapsulé (préfixé et/ou suffixé) par un entête de taille maximale η_L. La taille maximum d'un paquet circulant sur le lien L est donc $\eta_L + \psi_L = \mu_{f,L}$. En théorie des réseaux, on parle classiquement d'unité maximale de transmission, ou *Maximum Transmission Unit* (MTU). Enfin, le nombre maximal de paquets requis pour transporter une donnée du flux f après fragmentation est noté $\phi_{f,L}$, et la somme des tailles de ces paquets est notée $\theta_{f,L}$.

Ce modèle est illustré en Figure 2.5.

Le mécanisme d'ordonnancement peut imposer une fragmentation plus fine que ce que permet le protocole bas niveau, par exemple dans une optique de réduction des délais d'attente pire cas. Cette limite imposée par l'ordonnancement à la taille maximale de charge utile dans les paquets d'un flux f sur un lien L est notée $\psi'_{f,L}$. La taille maximale de charge utile pouvant être embarquée dans un paquet est donc égale au minimum entre la taille maximale imposée par le protocole et

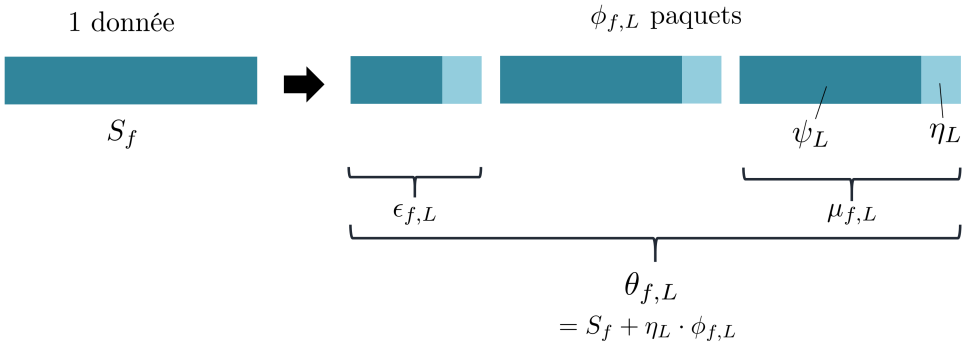

FIGURE 2.5 – Modèle de fragmentation

la limite imposée par le module d'ordonnancement pour le flux : $\min(\psi_L, \psi'_{f,L})$. En conséquence, un paquet du flux f sur le lien L ne peut excéder la taille de $\min(\psi_L, \psi'_{f,L}) + \eta_L$ bits.

Le nombre de paquets nécessaires à l'envoi de la donnée est égal à la borne entière supérieure du quotient entre la taille de la donnée brute et le nombre maximal de bits que l'on peut embarquer dans chaque paquet. On a donc :

$$\phi_{f,L} = \left\lceil \frac{S_f}{\min\left(\psi_L, \psi'_{f,L}\right)} \right\rceil \tag{2.2}$$

Ainsi, le nombre total de bits nécessaires à l'envoi de la donnée est :

$$\theta_{f,L} = S_f + \eta_L \cdot \phi_{f,L} = S_f + \eta_L \cdot \left\lceil \frac{S_f}{\min\left(\psi_L, \psi'_{f,L}\right)} \right\rceil \tag{2.3}$$

Enfin, la taille des paquets de tête du flux sur le lien L est égale à soit :

— la taille maximale d'un paquet si la taille totale de donnée est plus grande ;
— la taille totale de donnée si un seul paquet est suffisant pour l'envoyer.

On a donc :

$$\mu_{f,L} = \min\left(\psi_L, \psi'_{f,L}, S_f\right) + \eta_L \tag{2.4}$$

On en déduit la taille du paquet de queue :

$$\epsilon_{f,L} = \theta_{f,L} \% \mu_{f,L} \tag{2.5}$$

Par exemple, un flux f transmet des données de taille $S_f = 100\,\text{bit}$. Sur le lien, le protocole sous-jacent transmet des paquets d'entête de taille maximale $\eta_{LE} = 10\,\text{bit}$ et de charge utile de taille maximale $\psi_{LE} = 40\,\text{bit}$. Cependant, l'ordonnancement est configuré de telle sorte que le flux ne peut pas envoyer plus de $\psi'_{f,LE} = 30\,\text{bit}$ de charge utile, pour des raisons de lissage du trafic. Cette valeur limitera donc la taille de charge utile maximale.

Sur le lien, les données du flux seront fragmentées telles que sur la Figure 2.6. Les formules

décrites ci-dessus nous permettent d'obtenir directement les valeurs liées à la fragmentation :

$$\textit{Nombre de fragments} : \phi_{f,L} = \left\lceil \frac{100}{\min(40, 30)} \right\rceil = 4 \tag{2.6}$$

$$\textit{Taille totale} : \theta_{f,L} = 100 + 10 \cdot 4 = 140 \text{ bit} \tag{2.7}$$

$$\textit{Taille des paquets de tête} : \mu_{f,L} = \min(40, 30, 100) + 10 = 40 \text{ bit} \tag{2.8}$$

$$\textit{Taille du paquet de queue} : \epsilon_{f,L} = 140 \% 40 = 20 \text{ bit} \tag{2.9}$$

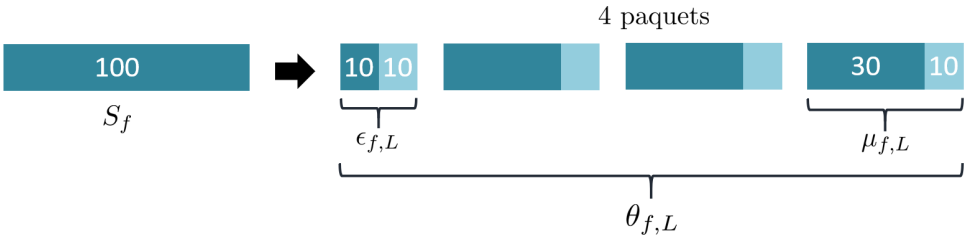

FIGURE 2.6 – Exemple de fragmentation

Délais liés à l'ordonnancement

Les paramètres protocolaires du flux permettent de garantir le respect de sa contrainte de bande passante. Le contrôleur utilisera cette information pour traiter la requête de flux, néanmoins il doit également être capable de garantir le respect de l'échéance pour chaque donnée du flux.

Le temps pire cas entre l'envoi du premier bit de la donnée et l'arrivée de son dernier bit sur le processeur de destination peut être calculé à partir de la combinaison des valeurs de délais suivantes :

— Le **délai de saut** est le délai pire cas subi et accumulé par la donnée à chaque saut, hors délai de commutation. Il est noté $d_{f,L}$;

— Le **délai de rafale** est le délai pire cas subi une fois par la rafale de donnée, sur le pire lien en termes de temps de transmission pire cas. Il est noté $\beta_{f,\mathcal{R}}$.

Le module d'ordonnancement doit préciser les fonctions ou algorithmes suivants, permettant de calculer ces valeurs de délais (Figure 2.7) :

$$d(f, L) = d : f, L \to d_{f,L} \tag{2.10}$$

$$\beta(f, \mathcal{R}) = \beta : f, \mathcal{R} \to \beta_{f,\mathcal{R}} \tag{2.11}$$

En résumé, l'interface entre les modules d'ordonnancement et contrôleur est composée des trois fonctions suivantes : fonction protocolaire, fonction de latence de saut et fonction de latence de rafale (Figure 2.7).

Synthèse des symboles

Les symboles définis dans cette section sont synthétisés en Tableau 2.1.

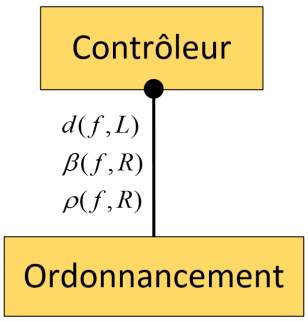

FIGURE 2.7 – Interface module d'ordonnancement/module contrôleur

TABLE 2.1 – Table des symboles

Symbole	Définition	Moyen mnémotechnique
T_f	Période minimale d'envoi de donnée du flux f	Symbole de période
S_f	Taille maximale de donnée du flux f	Size
dl_req_f	Échéance relative du flux f	Deadline requirement
C_L	Débit utile maximal sur le lien L	Capacité
$\mu_{f,L}$	Taille des paquets de tête d'une donnée du flux f sur le lien L	MTU
$\epsilon_{f,L}$	Taille du paquet de queue d'une donnée du flux f sur le lien L	Symbole de reliquat
ψ_L	Charge utile maximale sur le lien L	Payload size
$\psi'_{f,L}$	Limite à la charge utile pour le flux f sur le lien L	Payload size
η_L	Taille d'entête maximale sur le lien L	Header
$\phi_{f,L}$	Nombre maximal de fragments pour une donnée du flux f sur le lien L	Fragments
$\theta_{f,L}$	Taille totale d'une donnée du flux f sur le lien L après fragmentation	Total size
$\rho_{f,\mathcal{R}}$	Paramètres protocolaires du flux f dans le réseau $\mathcal{R}$	
$d_{sw,\mathcal{R}}$	Latence de commutation (constante)	
$d_{f,L}$	Délai de saut du flux f sur le lien L	
$\beta_{f,L}$	Délai de rafale du flux f sur le lien L	Burst delay

2.5.2 Module concret : virtualisation des liens

Ordonnancement Credit-Bounded Weighted Round Robin

Pour résoudre le problème d'ordonnancement des flux, nous proposons un usage dynamique de sous-canaux virtuels. Chaque lien L est divisé en Ch_L sous-canaux virtuels alloués aux flux. L'ensemble des sous-canaux virtuels alloués à un flux est appelé canal virtuel. Le nombre de sous-canaux virtuels composant un canal virtuel est appelé **poids du flux sur le lien**, noté $w_{f,L}$. Étant donné notre hypothèse d'homogénéité du réseau (cf. Section 2.3), chaque flux ne disposera que de deux valeurs de poids, l'une pour les liens internes ($w_{f,LI}$) et l'une sur les liens externes ($w_{f,LE}$). Ces deux valeurs constituent les paramètres protocolaires de notre ordonnanceur :

$$\rho_{f,\mathcal{R}} = \{w_{f,LI}, w_{f,LE}\} \tag{2.12}$$

L'ordonnancement utilisé est un ordonnancement basé sur des crédits et pondéré, *Credit-Bounded Weighted Round Robin* (CBWRR). Il se base sur l'ordonnancement *Deficit Weighted Round Robin* [Shreedhar and Varghese, 1996], qui a été montré comme approprié pour réaliser une allocation incrémentale de ressources réseau [Boyer et al., 2013]. Le principe est le suivant : chaque cycle, un nombre maximum de Q_L bits (le quantum) peut être envoyé par chaque sous-canal, chacun à leur tour. En conséquence, un maximum de $Q_L \cdot w_{f,L}$ bits d'un flux f peuvent être envoyés à chaque cycle par son canal virtuel. Un jeton passe de canal virtuel en canal virtuel pour les autoriser à émettre. S'il n'y a plus assez de crédit disponible pour envoyer le prochain paquet d'un canal virtuel, alors le jeton passe directement au suivant et le crédit restant est perdu ; le paquet sera envoyé au prochain cycle. La non conservation du crédit restant nous permet de mieux maîtriser l'isolation des flux et les latences pire cas. S'il n'y a aucune donnée à envoyer dans un canal virtuel, le jeton passe également au suivant. Dans un ordonnancement *Deficit Weighted Round Robin*, il est possible de conserver le crédit non consommé (le déficit) pour le prochain cycle. Puisque nous avons retiré cette fonctionnalité, le terme de déficit n'a plus lieu d'être dans le nom de l'ordonnanceur, c'est pourquoi nous l'avons renommé en *Credit-Bounded Weighted Round Robin* (CBWRR). La Figure 2.8 illustre le fonctionnement du CBWRR.

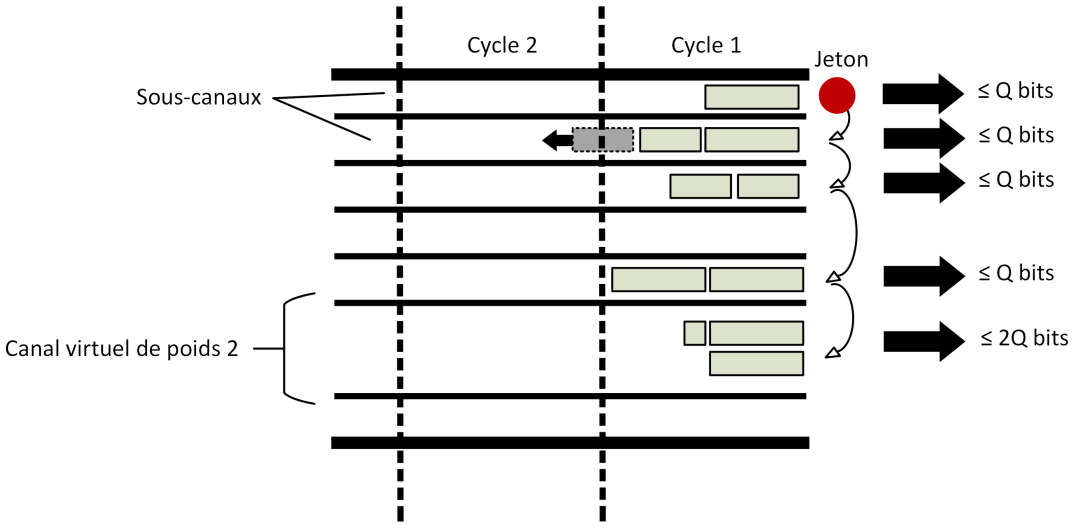

FIGURE 2.8 – Credit-Bounded Weighted Round Robin

CBWRR est similaire à une approche par multiplexage temporel (Time-Division Multiple

Access, TDMA), mais nous proposons une approche basée sur des crédits pour sa plus grande facilité de manipulation.

Enfin, nous proposons qu'un canal virtuel soit dédié aux flux non temps réel, c'est-à-dire utilisable sans contrôle mais sans aucune garantie. On parle de canal *best-effort*. Puisque le jeton du CBWRR saute tous les sous-canaux vides, il tournera plus vite lorsque les flux temps réel n'utilisent pas leur canal virtuel. Cela signifie que la bande passante non utilisée est automatiquement utilisable par le canal non temps réel, ce qui nous assure que l'utilisation de la bande passante est maximale. En revanche, lorsque les canaux virtuels temps réel ont besoin d'envoyer des paquets, le débit de sortie du canal virtuel non temps réel sera limité et maîtrisé.

Un exemple d'organisation des sous-canaux virtuels est donné en Figure 2.9.

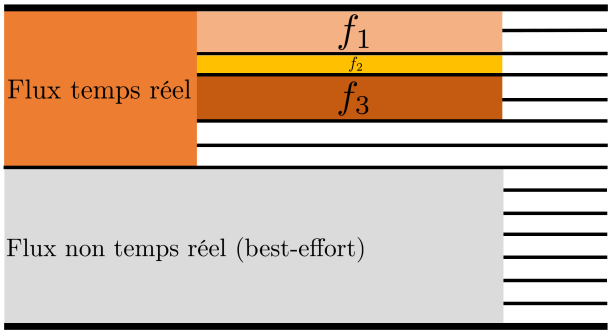

FIGURE 2.9 – Utilisation des sous-canaux virtuels avec criticité mixte

Propriétés du module

Comme défini en Section 2.5.1, le module d'ordonnancement doit garantir des propriétés d'isolation, déterminisme et équité suffisante.

Dans notre cas, le CBWRR est déterministe car son jeton tourne toujours dans le même sens. L'isolation des flux est garantie par le fait que le nombre de bits envoyés par chaque canal virtuel est limité. Ainsi, un flux de communication qui enverrait plus de données qu'initialement prévu ne peut pas interférer avec les autres. Son dysfonctionnement n'a d'effet que sur lui-même. Enfin, nous devons assurer que l'équité est suffisante pour pouvoir borner les latences et empêcher la famine. Dans le cas d'un canal virtuel de poids $w_{f,L}$, un flux dont les paquets seraient plus grands que $Q_L \cdot w_{f,L}$ ne pourrait jamais transmettre, car le crédit ne serait jamais suffisant en début de cycle, et perdu ensuite. En conséquence, il est nécessaire de limiter la taille des paquets par $Q_L \cdot w_{f,L}$. On a donc pour limite de charge utile :

$$\psi'_{f,L} = Q_L \cdot w_{f,L} - \eta_L \tag{2.13}$$

La propriété d'équité suffisante est ensuite garantie par le fait que :

1. Le canal virtuel du flux peut toujours envoyer au moins un paquet par cycle ;
2. La durée d'un cycle complet est limitée par le temps nécessaire pour envoyer $Q_L \cdot Ch_L$ bits, soit $Q_L \cdot Ch_L / C_L$ secondes.

Fonction protocolaire

Comme défini plus haut, les paramètres protocolaires de notre module concret d'ordonnancement sont les poids du flux sur les liens internes et externes. La fonction protocolaire doit donc

être capable de calculer le poids d'un flux sur un lien, tel que sa contrainte de bande passante est respectée.

On définit un réseau stable comme un réseau où le maximum de paquets de flux temps réel mis en tampon est borné. De même, on définit un canal virtuel stable comme un canal virtuel où le maximum de paquets temps réel est borné. Un réseau est stable si tous ses canaux virtuels sont stables. De plus, les contraintes de bande passante d'un flux sont respectées si tous les canaux virtuels correspondant à ce flux sont stables. L'objectif de notre fonction protocolaire est donc de calculer les poids minimum pour assurer que les canaux virtuels du flux sont stables.

En injectant la valeur de charge utile maximale $\psi'_{f,L} = Q_L \cdot w_{f,L} - \eta_L$, les équations 2.2 à 2.5 deviennent :

$$\textit{Nombre de fragments} : \phi_{f,L} = \left\lceil \frac{S_f}{\min\left(\psi_L, Q_L \cdot w_{f,L} - \eta_L\right)} \right\rceil \tag{2.14}$$

$$\textit{Taille totale} : \theta_{f,L} = S_f + \eta_L \cdot \phi_{f,L} \tag{2.15}$$

$$\textit{Taille des paquets de tête} : \mu_{f,L} = \min\left(\psi_L + \eta_L, Q_L \cdot w_{f,L}, \theta_{f,L}\right) \tag{2.16}$$

$$\textit{Taille du paquet de queue} : \epsilon_{f,L} = \theta_{f,L} \% \mu_{f,L} \tag{2.17}$$

On rappelle également que, selon nos hypothèses (cf. Section 2.3), le réseau est complètement fiable.

On définit ensuite :

— $\zeta_{f,L}$ la bande passante offerte maximale, en bits par cycle. Il s'agit de la bande passante maximale à laquelle peut réellement prétendre le flux sur le lien, en tenant compte des éventuelles pertes de crédit ;

— $bw\_req_{f,L}$ la bande passante requise, en bits par cycle. Elle sera calculée à partir des propriétés du flux $propriétés_f$.

Un canal virtuel est stable si $\zeta_{f,L}$ est supérieur ou égal à $bw\_req_{f,L}$. Le poids minimum requis sur le lien est donc :

$$w_{f,L} = \min_{w_{f,L} \in \mathbb{N}^+} \left(w_{f,L} | \zeta_{f,L} \geq bw\_req_{f,L}\right) \tag{2.18}$$

La bande passante offerte maximale est calculée en simulant le fait que le canal est chargé au maximum de paquets de taille $\theta_{f,L}$. De cette façon, on simule la bande passante maximale à laquelle le flux pourra prétendre. A chaque cycle, du crédit est perdu ou non. On continue jusqu'à atteindre le point de répétition du scénario, c'est-à-dire l'envoi du premier paquet d'une donnée en début de cycle. Ce point de répétition est appelé l'hyper-période et se mesure en nombre de cycles. Une fois atteinte, on divise le nombre total de bits qui ont pu être transmis par sa durée. Ce processus est réalisé par l'Algorithme 1 et illustré en Figure 2.10.

Notons que le principe de simuler l'ordonnancement jusqu'à l'hyper-période est utilisé en théorie de l'ordonnancement, et permet de construire des algorithmes dits exacts, comme par exemple dans [Fisher et al., 2006]. En effet, il a été montré pour plusieurs ordonnancements que le fait de les examiner sur un intervalle de temps dépendant de l'hyper-période permet de démontrer leur faisabilité ([Leung and Merrill, 1980] et [George et al., 1996]). Cependant, l'hyper-période correspond dans ce cas au plus petit commun multiple entre les périodes des ensembles de tâches. Il s'agit donc d'un point de répétition du scénario en termes d'arrivées de tâches, tandis que, dans notre cas, on s'intéresse à la répétition du scénario d'utilisation du canal virtuel.

Algorithme 1 : Calcul de la bande passante offerte maximale

```
1  Fonction bande_passante_offerte_maximale(w_{f,L}, Q_L, ψ_L, η_L, θ_{f,L}, φ_{f,L})
2      donnée_fragmentée : Liste ;
3      μ_{f,L} ← min(ψ_L + η_L, Q_L * w_{f,L}, θ_{f,L}) ;
4      ε_{f,L} ← θ_{f,L} % μ_{f,L} ;
       /* Construction du modèle de donnée fragmentée                    */
5      pour k de 0 à φ_{f,L} − 1 faire
6          donnée_fragmentée.ajout(μ_{f,L}) ;
7      fin
8      donnée_fragmentée.ajout(ε_{f,L}) ;
       /* Calcul de la bande passante offerte maximale                   */
9      cycle ← 1 ;
10     curseur ← 0 ;
11     total_envoyé ← 0 ;
12     tant que vrai faire
13         crédit ← Q_L * w_{f,L} ;
14         tant que crédit ≥ donnée_fragmentée.accéder(curseur) faire
15             crédit ← crédit - donnée_fragmentée.accéder(curseur) ;
16             total_envoyé ← total_envoyé + donnée_fragmentée.accéder(curseur) ;
17             curseur ← (curseur+1)% donnée_fragmentée.taille() ;
18         fin
           /* Hyper-période                                                 */
19         si curseur == 0 alors
20             retourner total_envoyé/cycle ;
21         fin
22         cycle++ ;
23     fin
```

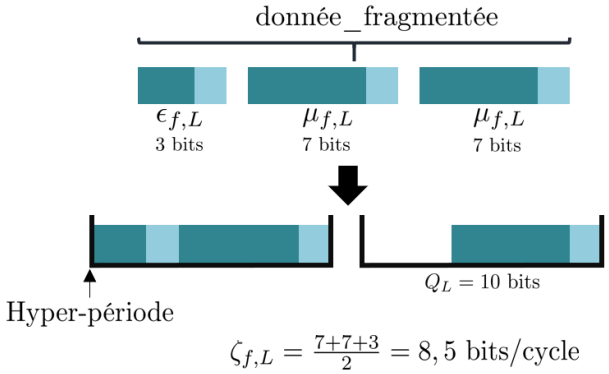

FIGURE 2.10 – Illustration du calcul de la bande passante offerte maximale

La bande passante requise est le nombre moyen de bits par cycle que le flux émet dans le pire cas, en tenant compte de la surcharge due à la fragmentation et l'encapsulation des fragments. Dans le pire cas, le flux émet une donnée de taille maximale S_f toutes les périodes minimales T_f. Sa bande passante requise en bits par cycle est donc égale au quotient entre la taille maximale de donnée fragmentée $\theta_{f,L}$ par la période minimale en cycles, $T_f c$.

Un lien émet un maximum de $Q_L \cdot Ch_L$ bits par cycle, la durée maximale d'un cycle est donc de $Q_L \cdot Ch_L / C_L$ secondes. On en déduit :

$$T_f c = \frac{T_f}{\frac{Q_L \cdot Ch_L}{C_L}} = \frac{T_f \cdot C_L}{Ch_L \cdot Q_L} \tag{2.19}$$

Ainsi :

$$bw\_req_{f,L} = \frac{\theta_{f,L}}{T_f c} = \frac{\theta_{f,L}}{\frac{T_f \cdot C_L}{Ch_L \cdot Q_L}} = \frac{\theta_{f,L} \cdot Ch_L \cdot Q_L}{T_f \cdot C_L} \tag{2.20}$$

En combinant les équations 2.14, 2.15, 2.18, 2.20 et l'Algorithme 1, on obtient le théorème suivant, permettant de calculer les paramètres protocolaires $w_{f,LI}$ et $w_{f,LE}$ d'un flux.

Théorème 1. *Un réseau SDRN sur CBWRR est stable si, pour chaque flux temps réel (PRT) $f \in \mathcal{F}_\mathcal{R}$, les paramètres protocolaires $w_{f,LI}$ et $w_{f,LE}$ obéissent :*

$$\min_{w_{f,L} \in \mathbb{N}^+} \left(w_{f,L} \middle| \zeta_{f,L} \geq \frac{\theta_{f,L} \cdot Ch_L \cdot Q_L}{T_f \cdot C_L} \right) \tag{2.21}$$

avec :
η_L = *taille d'entête maximale sur le lien* L
ψ_L = *taille de charge utile maximale sur le lien* L
$\phi_{f,L} = \lceil S_f / \min(w_{f,L} \cdot Q_L - \eta_L, \psi_L) \rceil$
$\theta_{f,L} = S_f + \eta_L \cdot \phi_{f,L}$
$\zeta_{f,L}$ = *bande passante offerte maximale, calculée par l'Algorithme 1*

L'utilisation de ce théorème est effectuée par l'Algorithme 2. En résumé, on débute avec un poids de 1, on calcule les bandes passantes offertes et requises, puis on vérifie si la première est supérieure ou égale à la deuxième. Si oui, on a trouvé le poids minimum requis. Si non, on incrémente le poids et on recommence. Si l'on a dépassé le nombre total de sous-canaux Ch_L, alors il n'y a pas de solution, le flux ne peut être alloué sur le réseau. Cela signifie qu'il y a eu une erreur de conception, car le flux ne peut même pas être alloué sur un réseau vide.

Illustrons le processus avec un exemple simple :
- Taille de donnée $S_f = 100\,\text{bit}$;
- Période minimale $T_f = 12\,\mu\text{s}$;
- Taille d'entête maximale $\eta_L = 10\,\text{bit}$;
- Charge utile maximale protocolaire $\psi_L = 40\,\text{bit}$;
- Débit utile $C_L = 1\,\text{Gbit/s}$;
- Nombre de sous-canaux virtuels $Ch_L = 100$;
- Quantum $Q_L = 10\,\text{bit}$;

En débutant avec un poids de 1 que l'on incrémente, on obtient les valeurs du Tableau 2.2. Lorsque $w_{f,L} = 2$, la bande passante offerte devient supérieure à la bande passante requise. Il s'agit donc du poids minimum requis pour disposer d'un canal virtuel stable pour le flux f sur le lien L.

On peut également faire l'observation suivante. Lorsque le poids vaut 1, la valeur de charge utile maximale protocolaire ψ_L est limitée par le nombre maximum de bits que le canal virtuel

Algorithme 2 : Calcul du poids du flux

```
1  Fonction calculer_w_{f,L}(requête_f, ψ_L, η_L, Q_L, Ch_L, C_L)
2      w_{f,L} ← 1 ;
3      tant que w_{f,L} ≤ Ch_L faire
4          calculer φ_{f,L} ;
5          calculer θ_{f,L} ;
6          calculer ζ_{f,L} ;
7          si ζ_{f,L} ≥ θ_{f,L} · Ch_L · Q_L/T_f · C_L alors
8              retourner w_{f,L} ;
9          fin
10         sinon
11             w_{f,L} ← w_{f,L} + 1 ;
12         fin
13     fin
14     retourner −1 ;
```

TABLE 2.2 – Illustration de l'analyse du poids minimum requis

	$w_{f,L} = 1$	$w_{f,L} = 2$
ψ_L	40 bit	40 bit
$\psi'_{f,L} = Q_L \cdot w_{f,L} + \eta_{f,L}$	30 bit	70 bit
$\phi_{f,L}$	4 paquets	3 paquets
$\theta_{f,L}$	140 bit	130 bit
$\zeta_{f,L}$	35 bit/cycle	46,67 bit/cycle
$bw\_req_{f,L}$	35 bit/cycle	43,34 bit/cycle
$\zeta_{f,L} \geq bw\_req_{f,L}$	faux	vrai

peut envoyer chaque cycle, $\psi'_{f,L}$. Lorsque le poids vaut 2, ce n'est plus le cas. La donnée est donc fragmentée en moins de paquets, ce qui réduit également le surcoût dû à un nombre réduit d'entêtes, et donc la bande passante requise.

Fonction de délai de saut

Le délai de saut $d_{f,L}$ est un délai pire cas accumulable et subi à chaque saut de bout-en-bout. Dans un premier temps, nous allons calculer le délai pire cas que peut subir le premier paquet d'une donnée fragmentée temps réel (flux de type PRT), puis nous en déduirons le délai de saut.

Le temps pire cas entre le moment où le premier paquet d'une donnée PRT fragmentée a été placé dans son tampon de sortie et le moment où il a été complètement transmis au nœud suivant peut être décomposé en deux sous-délais :

— le temps d'attente pire cas avant le début de la transmission ;
— le temps de transmission du paquet.

Ce paquet est de taille $\mu_{f,L}$ (équation 2.16), on a donc :

$$d_{f,L} = \frac{\mu_{f,L}}{C_L} + attente_{f,L} \tag{2.22}$$

On donne ensuite le théorème suivant :

Théorème 2. *Dans un réseau SDRN sur CBWRR stable, le délai d'attente pour le premier paquet d'une donnée PRT fragmentée, qui a été placé dans le tampon de sortie de son canal virtuel et attend le début de sa transmission, ne peut excéder* $(Ch_L - w_{f,L}) \cdot Q_L / C_L$ *secondes. De plus, sur le pire lien en termes de temps de transmission, le temps entre l'arrivée d'une donnée PRT fragmentée et la fin de sa transmission ne peut excéder* $\left(\frac{\theta_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L / \mu_{f,L} \rfloor \cdot \mu_{f,L}} \right\rceil \cdot \frac{(Ch_L - w_{f,L}) \cdot Q_L}{C_L} \right)$ *secondes.*

Démonstration. On définit :

— $t_{i,a}$ le moment, en secondes, où le premier paquet de la $i^{ème}$ donnée fragmentée du flux PRT est arrivé dans le canal virtuel (le premier bit a été placé dans le tampon) ;
— $t_{i,s}$ le moment, en secondes, où le premier paquet de la $i^{ème}$ donnée fragmentée du flux PRT a commencé à être transmis ;
— $t_{i,f}$ le moment, en secondes, où le dernier paquet de la $i^{ème}$ donnée fragmentée du flux PRT a fini d'être transmis ;
— $x = (Ch_L - w_{f,L}) \cdot Q_L / C_L$, le temps maximum pendant lequel on peut attendre le retour du jeton d'ordonnancement CBWRR.

On veut prouver que $P(i) : \forall i \in \mathbb{N}, t_{i,s} - t_{i,a} \leq x$ et $t_{i,f} - t_{i,a} \leq \frac{\theta_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L / \mu_{f,L} \rfloor \cdot \mu_{f,L}} \right\rceil \cdot x$ est vraie. A cette fin, on utilise un raisonnement par récurrence.

Initialisation :

Pour le tout premier paquet du flux, le temps d'attente pire cas apparaît lorsque le jeton passe au canal suivant au moment où le paquet est placé dans le tampon, et qu'il faut attendre son retour avant de commencer la transmission : $t_{1,s} - t_{1,a} = x$. On suppose ensuite qu'on étudie le pire lien (en termes de temps de transmission). Dans ce goulot d'étranglement, les paquets d'une

même donnée fragmentée se suivent dans le canal virtuel, sans intervalle entre la fin de la transmission d'un paquet et le début de la transmission du suivant du point de vue du canal virtuel [2]. Transmettre une donnée fragmentée de taille $\theta_{f,L}$ prend $\theta_{f,L}/C_L$ secondes de pure transmission. De plus, chaque cycle, le canal virtuel pourra transmettre au plus $\lfloor w_{f,L} \cdot Q_L/\mu_{f,L} \rfloor$ paquets de taille maximum $\mu_{f,L}$, éventuellement suivis par le paquet de queue. Cela signifie que, dans le pire cas, le canal virtuel devra attendre le retour du jeton d'ordonnancement $\left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L/\mu_{f,L} \rfloor \cdot \mu_{f,L}} - 1 \right\rceil$ fois avant d'avoir transmis l'ensemble des paquets correspondant à la donnée fragmentée.

On a donc :

$t_{1,f} - t_{1,s} = \frac{\theta_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L/\mu_{f,L} \rfloor \cdot \mu_{f,L}} - 1 \right\rceil \cdot x$

$t_{1,f} - t_{1,a} = \frac{\theta_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L/\mu_{f,L} \rfloor \cdot \mu_{f,L}} \right\rceil \cdot x$

Donc $P(1)$ est vraie.

Hérédité :

On suppose $P(i)$ est vraie et étudie la $(i+1)^{ème}$ donnée fragmentée.

Le pire cas, à la fois en termes de temps d'attente et de délai total, est quand la $(i+1)^{ème}$ donnée fragmentée n'utilise pas le crédit restant du cycle en cours et attend le prochain pour commencer sa transmission : $t_{i+1,s} - t_{i,f} = x$. Dans ce cas, le motif de transmission est le même que celui de la première donnée fragmentée du flux : $t_{i+1,f} - t_{i+1,s} = \frac{\theta_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L/\mu_{f,L} \rfloor \cdot \mu_{f,L}} - 1 \right\rceil \cdot x$

Dans ce pire cas où le motif de transmission est le même pour chaque donnée fragmentée, l'hypothèse de stabilité du réseau implique que $t_{i+1,a} \geq t_{i,f}$.

Ainsi, dans le pire cas :

$t_{i+1,f} - t_{i+1,a} \leq (t_{i+1,f} - t_{i+1,s}) + (t_{i+1,s} - t_{i,f})$

$t_{i+1,f} - t_{i+1,a} \leq \frac{\theta_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L/\mu_{f,L} \rfloor \cdot \mu_{f,L}} \right\rceil \cdot x$

Donc $P(i+1)$ est vraie si $P(i)$ est vraie.

Conclusion :

D'après le principe de récurrence, $P(i)$ est vraie $\forall i \in \mathbb{N} \setminus 0$. □

Le Théorème 2 nous permet de calculer le temps d'attente pire cas du premier paquet de la donnée fragmentée. Les paquets suivants de la même donnée font apparaître l'un des deux cas suivants lorsqu'ils sont placés dans le tampon de sortie :

- **Cas 1** : il y a d'autres paquets dans le canal virtuel dont la transmission est en cours ou imminente. Dans ce cas, le délai d'attente cumulé n'augmente pas car le canal virtuel est constamment en train de transmettre lorsqu'il possède le jeton d'ordonnancement, il n'y a donc pas d'attente ;
- **Cas 2** : le canal virtuel est vide. Cela implique qu'il n'y a aucune attente supplémentaire due à la présence d'autres paquets dans le tampon. Dans ce cas, le délai d'attente cumulé est remis à zéro et sa valeur pire cas est à nouveau de $(Ch_L - w_{f,L}) \cdot Q_L/C_L$ secondes.

2. Notons qu'un espace inter-trames pourrait également être considéré en augmentant simplement la taille des paquets fragmentés considérés.

En conséquence, le délai d'attente cumulé pire cas pour l'ensemble de la donnée est égal à celui de son premier paquet.

L'équation 2.22 devient alors :

$$\begin{aligned} d_{f,L} &= \frac{\mu_{f,L}}{C_L} + \frac{(Ch_L - w_{f,L}) \cdot Q_L}{C_L} \\ &= \frac{\mu_{f,L} + (Ch_L - w_{f,L}) \cdot Q_L}{C_L} \end{aligned} \tag{2.23}$$

On en déduit la fonction de délai de saut, qui est un élément à fournir de l'interface ordonnancement/contrôleur (Figure 2.7) :

$$d(f, L) = d : f, L \rightarrow \frac{\mu_{f,L} + (Ch_L - w_{f,L}) \cdot Q_L}{C_L} \tag{2.24}$$

Fonction de délai de rafale

Le délai de rafale est le délai qui ne sera subi qu'une seule fois de bout-en-bout. L'interférence des paquets d'un flux entre eux ainsi que le temps de transmission du premier paquet de la donnée sont déjà inclus dans la fonction de délai de rafale. Le délai de rafale correspond donc au temps de transmission du reste de la donnée sur le pire lien en termes de temps de transmission (goulot d'étranglement). Compte tenu de nos hypothèses de recherche (cf. Section 2.3), il est égal au maximum de ce temps de transmission sur lien interne et sur lien externe. Pour le calculer, on utilise le Théorème 2 et retranchons le temps de transmission du premier paquet de la donnée, $\mu_{f,L}/C_L$. On en déduit la fonction de délai de rafale pour l'interface ordonnancement/contrôleur :

$$\beta(f, \mathcal{R}) = \beta : f, \mathcal{R} \rightarrow \max_{\{LI,LE\}} \left(\frac{\theta_{f,L} - \mu_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L / \mu_{f,L} \rfloor \cdot \mu_{f,L}} - 1 \right\rceil \cdot \frac{(Ch_L - w_{f,L}) \cdot Q_L}{C_L} \right) \tag{2.25}$$

Représentation sur un modèle de calcul réseau

Nous pouvons représenter le comportement d'un canal virtuel sur un modèle de calcul réseau, tel que présenté en Chapitre 1. Nous prenons un exemple simple correspondant à l'envoi d'une donnée sur un seul saut.

La courbe d'arrivée est une fonction affine. Son ordonnée à l'origine est la rafale maximale, c'est-à-dire la taille maximale d'une donnée fragmentée. Sa pente est, dans le pire cas, équivalente à l'envoi d'une donnée de taille maximale à chaque période minimale. La courbe de service est représentée par une version légèrement pessimiste de la réalité. Le délai borné est égal au temps d'attente pire cas avant la transmission du premier bit d'une donnée. Nous l'avons calculé plus haut. Concernant la pente, nous prenons une version légèrement pessimiste et considérons que le canal virtuel n'envoie que des paquets de tête. En combinant cette hypothèse avec la durée d'un cycle calculée plus tôt dans ce chapitre, nous obtenons une pente de courbe de service strictement inférieure à la courbe réelle. Les résultats de cette modélisation sont donnés en Figure 2.11.

Premièrement, nous pouvons constater que, le canal virtuel étant stable, l'écart entre les deux demi-droites ne peut excéder l'écart initial. On peut donc calculer la taille de file d'attente maximale et le délai pire cas, comme indiqués sur la Figure 2.11. La première est égale à la somme de la rafale et de la hauteur supplémentaire notée **A** sur la Figure 2.11. Cette hauteur est égale au produit de l'abscisse et de la pente de la courbe d'arrivée. On a donc :

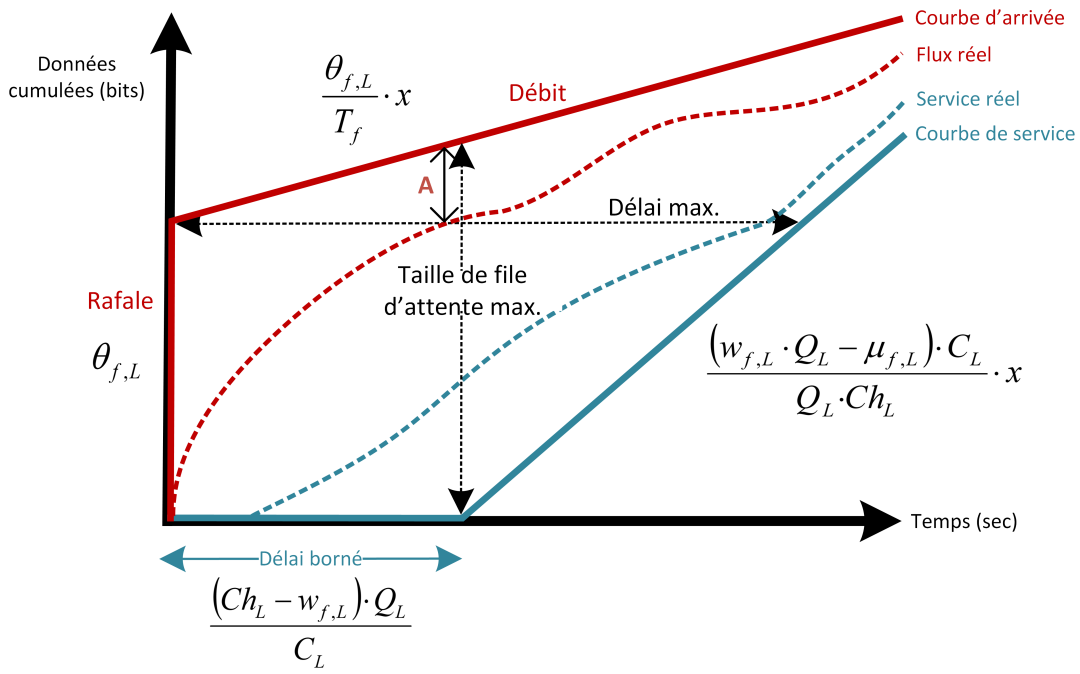

FIGURE 2.11 – Courbes d'arrivée et de service

$$\textit{Taille de file d'attente max.} = \theta_{f,L} + \frac{\theta_{f,L} \cdot (Ch_L - w_{f,L}) \cdot Q_L}{T_f \cdot C_L} \tag{2.26}$$

Deuxièmement, nous allons comparer le délai obtenu par ce modèle avec le délai obtenu en utilisant les formules définies plus tôt. Nous prenons l'exemple simple de l'envoi d'une donnée sur un saut, avec les caractéristiques suivantes : $C_L = 1\,\text{Mbit/s}$, $Ch_L = 100$, $Q_L = 1000\,\text{bit}$, $\theta_{f,L} = 1000\,\text{bit}$, $w_{f,L} = 1$. On considère que la donnée est constituée de trois paquets de tailles respectives 400 bit, 400 bit et 200 bit.

Si l'on calcule la distance horizontale représentée sur la Figure 2.11, on obtient un délai pire cas de 0,27 s. Si l'on calcule $d_{f,L} + \beta_{f,\mathcal{R}}$ en combinant les équations 2.23 et 2.25, on obtient un délai de 0,20 s. On constate que la valeur obtenue par calcul réseau est légèrement supérieure à la seconde valeur, ce qui était attendu puisque la courbe de service est pessimiste par rapport à notre méthode d'analyse détaillée plus haut. La valeur obtenue par calcul réseau pourrait donc être réduite avec ce pessimisme.

Cet exemple simple nous permet de montrer qu'il est possible d'obtenir des valeurs de délai similaires en s'appuyant sur un modèle de réseau, mais que la construction de ce modèle nécessite une analyse préliminaire telle que celle que nous avons conduite dans ce chapitre. Ce modèle nous permet également de calculer facilement les tailles de tampons nécessaires pour éviter les débordements. Ceci n'est évidemment valable que pour les canaux virtuels stables. Pour les canaux non temps réel ne proposant aucune garantie, les tailles de tampons doivent être décidées par le concepteur du système, en fonction des différents flux de communication et du taux d'acceptation des débordements.

2.5.3 Synthèse

Le module d'ordonnancement est le mécanisme de gestion des flux en sortie de nœud, en surcharge du protocole sous-jacent de liaison de données. Il doit respecter trois propriétés pour que le paradigme SDRN puisse être appliqué par-dessus : isolation, déterminisme et équité minimale. Il peut utiliser des paramètres protocolaires supplémentaires assignés au flux, telles que des valeurs dans des champs de l'entête protocolaire sous-jacent (valeurs de poids, de priorité, de classe de trafic, etc.). Son interface avec le contrôleur SDRN est composé de trois fonctions : la fonction protocolaire permettant de déterminer les paramètres protocolaires du flux étudié sur chaque lien en fonction de ses contraintes temps réel, et les fonctions de délai de saut et de rafale utilisées pour l'analyse temporelle du flux.

Nous proposons l'utilisation d'un ordonnancement de type *Credit-Bounded Weighted Round Robin* en tant que module concret. Cet ordonnancement respecte les propriétés ci-dessus et permet une cohabitation douce entre flux temps réel et non temps réel, ainsi qu'entre flux de caractéristiques différentes. Il possède également l'avantage de permettre aux flux non temps réel de profiter automatiquement de la bande passante non utilisée. Ceci maximise l'utilisation de la bande passante de façon sécurisée d'un point de vue temps réel, sans besoin de contrôle extérieur supplémentaire. Enfin, nous avons défini l'interface entre ce module et le module contrôleur (Figure 2.12).

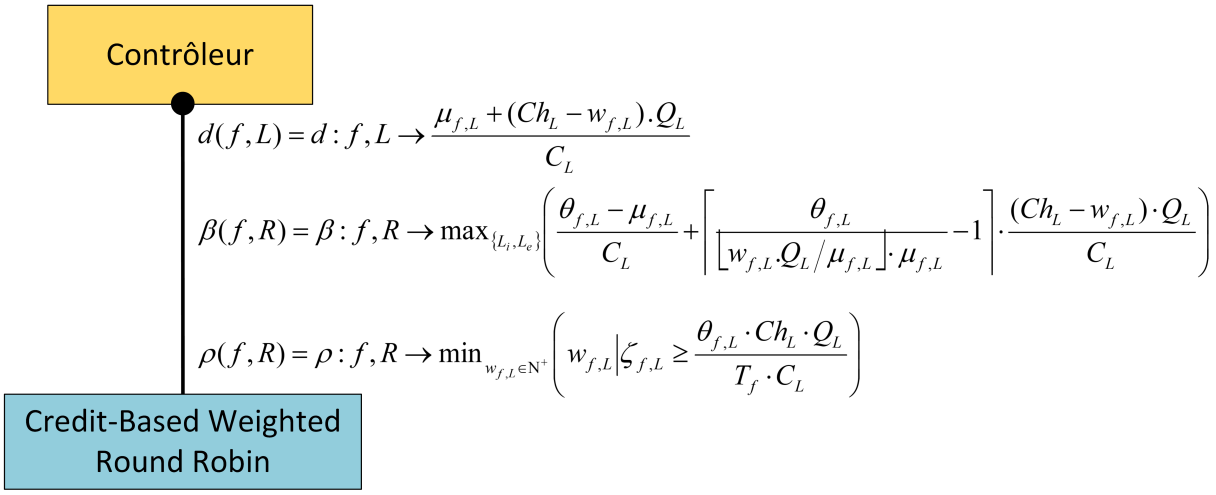

FIGURE 2.12 – Module concret d'ordonnancement

2.6 Contrôle d'admission et recherche de chemin : module contrôleur

2.6.1 Rôle du contrôleur

Le contrôleur est le chef d'orchestre du réseau SDRN. Sa responsabilité est de configurer le réseau en fonction des requêtes de flux transmises par les applications, ainsi que des informations sur les propriétés et l'état du réseau. L'interface avec le module *Propriétés liens et protocole* (Figure 2.4) lui fournit les informations sur les débits et modèles de fragmentation des liens internes et externes. L'interface avec le module d'ordonnancement (Figure 2.7) lui fournit les fonctions de calcul des paramètres protocolaires et valeurs de délai pire cas relatives à un flux.

Ces fonctions seront utilisées pour décider s'il est possible de garantir le respect des contraintes temps réel du flux étudié.

Dans un réseau SDRN, le contrôleur répond à la fois aux problématiques de routage et d'analyse temporelle des flux, au sein du même processus, le **contrôle d'admission**. Lorsque le contrôleur reçoit une requête de flux de la part d'une application, il réalise le contrôle d'admission de ce flux. Cela consiste à décider s'il existe un chemin et des paramètres protocolaires permettant au flux de respecter ses contraintes temps réel (bande passante et latence) en toute circonstance, et si oui lesquels. En d'autres termes, le contrôle d'admission SDRN correspond à une recherche de chemin sous contraintes.

On rappelle que le SDRN est basé sur une allocation entièrement incrémentale des flux de communication. En conséquence, le contrôle d'admission est réalisé sans connaissance des futures requêtes ; c'est une approche dite agnostique ou non-clairvoyante. Cela signifie qu'une décision de contrôle d'admission peut apparaître comme idéale à un instant donné, mais rendre l'allocation de futurs flux moins optimisée voire impossible. Pour cette raison, une stratégie d'optimisation doit être prise en compte lors du contrôle d'admission. Cette stratégie définit comment préparer au mieux l'avenir lors de l'allocation d'un flux.

Dans cette partie, nous allons décrire la solution de contrôle d'admission et recherche de chemin que nous proposons, puis en déduirons l'architecture du module contrôleur abstrait.

2.6.2 Modèle et contraintes

Comme défini en Section 2.2.1, le réseau est représenté sous la forme d'un graphe $\mathcal{G}_{\mathcal{R}}$, dont les sommets sont les nœuds et les arcs sont les liens internes et externes. Le contrôleur doit chercher un chemin sur ce graphe, selon trois contraintes dont les deux premières sont obligatoires :

1. Bande passante ;
2. Latence pire cas ;
3. Optimisation, préparation à l'avenir.

On rappelle en Tableau 2.3 les symboles définis précédemment ainsi que dans cette section.

Bande passante

La contrainte de bande passante est respectée si les paramètres protocolaires du flux peuvent être respectés sur tous les liens du chemin. Dans le cas du SDRN sur CBWRR, les paramètres protocolaires des flux sont leurs poids, c'est-à-dire le nombre minimal de sous-canaux que l'on doit leur allouer pour garantir que la contrainte de bande passante sera toujours respectée. Un chemin est donc valide en termes de bande passante s'il possède suffisamment de sous-canaux sur chacun des liens le composant. Afin d'avoir cette information, nous étiquetons les arcs du graphe avec le nombre restant de sous-canaux, R_L (Figure 2.13). A l'initialisation, on a $R_L = Ch_L$. Lorsqu'un flux est alloué sur un lien, R_L est diminué du poids de ce flux. Lorsqu'il est libéré, R_L est augmenté d'autant.

Garantir le respect de cette contrainte dans l'algorithme de chemin est donc relativement simple. On calcule les poids minimaux $w_{f,L}$ grâce à la fonction protocolaire $\rho(f, L)$ (équation 2.18). On retire ensuite tous les arcs du graphe satisfaisant $R_L < w_{f,L}$ avant d'effectuer la recherche de chemin. Ainsi, on est assuré que tout chemin dans le graphe respectera la contrainte de bande passante.

Cette méthode est classiquement utilisée en routage avec des contraintes de Qualité de Service. Dans ce domaine, on dit qu'une contrainte est non-additive lorsqu'elle doit être respectée

TABLE 2.3 – Table des symboles

Symbole	Définition	Moyen mnémotechnique
T_f	Période minimale d'envoi de donnée du flux f	Symbole de période
S_f	Taille maximale de donnée du flux f	Size
dl_req_f	Échéance relative du flux f	Deadline requirement
C_L	Débit utile maximal sur le lien L	Capacité
Ch_L	Nombre de sous-canaux virtuels sur le lien L	Channels
R_L	Nombre de sous-canaux virtuels vacants sur le lien L	Remaining
$\mu_{f,L}$	Taille des paquets de tête d'une donnée du flux f sur le lien L	MTU
$\epsilon_{f,L}$	Taille du paquet de queue d'une donnée du flux f sur le lien L	Symbole de reliquat
ψ_L	Charge utile maximale sur le lien L	Payload size
$\psi'_{f,L}$	Limite à la charge utile pour le flux f sur le lien L	Payload size
η_L	Taille d'entête maximale sur le lien L	Header
$\phi_{f,L}$	Nombre maximal de fragments pour une donnée du flux f sur le lien L	Fragments
$\theta_{f,L}$	Taille totale d'une donnée du flux f sur le lien L après fragmentation	Total size
$w_{f,L}$	Poids minimum requis du flux f sur le lien L	Weight
$d_{sw,\mathcal{R}}$	Latence de commutation (constante)	
$d_{f,L}$	Délai de saut du flux f sur le lien L	
$\beta_{f,L}$	Délai de rafale du flux f sur le lien L	Burst delay

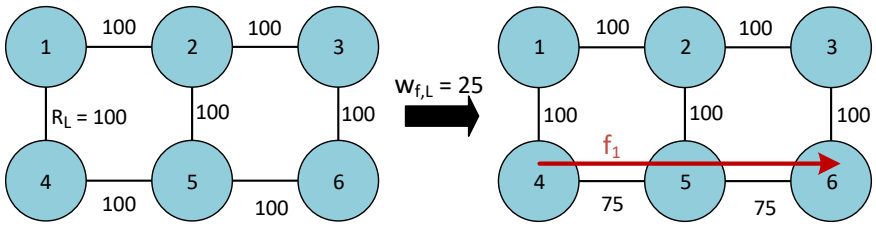

FIGURE 2.13 – Exemple d'évolution du nombre restant de sous-canaux

individuellement sur chacun des liens et ne se cumule pas [Korkmaz and Krunz, 2001]. C'est le cas notamment de la bande passante. Pour gérer une telle contrainte non-additive, la méthode usuelle est de retirer les liens qui ne la satisfont pas [Wang and Crowcroft, 1995], ce que nous faisons ici.

Délai pire cas

Nous proposons de traduire la contrainte de délai du flux en taille maximale de chemin, c'est-à-dire le nombre de nœuds le composant. Cette méthode, appelée discrétisation du délai, permet de réduire la complexité de la recherche de chemin [Chen et al., 2008]. La taille de chemin est notée $\|path_f\|$. Par exemple, le chemin, $path_f = \{N_4, N_5, N_6\}$ est de taille $\|path_f\| = 3$. Cette traduction est possible grâce à l'hypothèse d'homogénéité du réseau (cf. Section 2.3). En effet, les comportements des liens sont identiques entre eux dans le pire cas, c'est-à-dire dans le cas d'un réseau complètement chargé. Puisque nous nous intéressons toujours au pire cas, un saut sur un lien est équivalent à un saut sur un autre lien du même type, interne ou externe.

Pour atteindre son processeur de destination, la donnée fragmentée doit :

— effectuer deux sauts sur des liens internes : entre le processeur source et le commutateur du nœud source et entre le commutateur du nœud destination et le processeur destination ;
— effectuer $\|path_f - 1\|$ sauts sur des liens externes ;
— être commutée $\|path_f\|$ fois.

Le délai de saut est subi à chaque saut, tandis que le délai de rafale n'est subi qu'une fois de bout-en-bout. Enfin, le délai de commutation est une constante notée $d_{sw,\mathcal{R}}$ (cf. Section 2.2.3).

Soit $wc\_délai(f, path_f)$ le délai pire cas subi par une donnée fragmentée du flux f suivant le chemin $path_f$. Ce délai correspond à la différence de temps entre le moment où le premier bit de la donnée est envoyé par le processeur source et le moment où son dernier bit est reçu par le processeur destination.

On a donc :

$$wc\_délai(f, path_f) = 2 \cdot d_{f,LI} + (\|path_f\| - 1) \cdot d_{f,LE} + \|path_f\| \cdot d_{sw,\mathcal{R}} + \beta_{f,\mathcal{R}} \tag{2.27}$$

Un chemin $path_f$ respecte la contrainte de délai si le délai bout-en-bout pire cas est inférieur ou égal à la contrainte d'échéance, soit si $wc\_délai(f, path_f) \leq dl\_req_f$. Depuis cette contrainte et l'équation 2.27, on peut isoler $\|path_f\|$ pour en déduire la contrainte sur la taille du chemin :

$$\|path_f\| \leq \frac{dl\_req_f + d_{f,LE} - 2 \cdot d_{f,LI} - \beta_{f,\mathcal{R}}}{d_{sw,\mathcal{R}} + d_{f,LE}} \tag{2.28}$$

Ou encore :

$$taille\_max_f = \frac{dl\_req_f + d_{f,LE} - 2 \cdot d_{f,LI} - \beta_{f,\mathcal{R}}}{d_{sw,\mathcal{R}} + d_{f,LE}} \tag{2.29}$$

Un chemin est valide seulement s'il vérifie l'inéquation 2.28.

Stratégie d'optimisation

La stratégie d'optimisation définit la façon de choisir le chemin final parmi tous les chemins valides, c'est-à-dire respectant les contraintes de bande passante et de latence. Elle dépend des objectifs fixés par le concepteur du système. En voici quelques exemples :

— effectuer le contrôle d'admission au plus vite, donc prendre le plus court chemin ou le premier chemin valide trouvé ;

— concentrer les flux pour optimiser la consommation d'énergie ;
— répartir la charge ;
— optimiser l'usure des composants ;
— etc.

Cette stratégie d'optimisation peut nécessiter une représentation sur le modèle du réseau, sous la forme d'un ou plusieurs critère(s) d'optimisation. Nous proposons d'utiliser le coût des arcs du graphe pour représenter ces critères d'optimisation. Moins le chemin est coûteux, plus il est optimisé. L'algorithme de recherche de chemin doit donc chercher le chemin le moins coûteux parmi ceux qui sont suffisamment courts pour satisfaire l'équation 2.29.

Dans notre cas, nous utilisons un critère d'optimisation lié à la charge des liens. L'objectif est de répartir la charge sur le réseau de sorte à préparer l'avenir. En effet, si l'on se contente d'allouer les flux sur le plus court chemin, il y a des risques de surcharger certaines zones du réseau. Ceci peut rendre impossible l'allocation d'un futur flux possédant une contrainte d'échéance serrée. La répartition de la charge permet de réduire ce risque : les flux peuvent être alloués sur des chemins plus longs si cela permet d'alléger des parties du réseau. Notons que plus le système de communication est composé de flux volumineux, c'est-à-dire dont le nombre de sous-canaux requis est élevé, moins cette stratégie est efficace. En effet, répartir la charge risque de rendre l'allocation d'un flux volumineux impossible car il n'existe plus assez de liens suffisamment peu chargés pour l'accepter. Une allocation plus déséquilibrée serait plus intéressante dans ce cas. Pour ces raisons, il est important de garder à l'esprit que la stratégie d'optimisation doit être choisie en fonction du système et de ses objectifs. La stratégie de répartition de charge que nous proposons est un exemple suffisamment générique pour nous permettre de mettre en avant l'utilisation du critère d'optimisation, tout en offrant une première optimisation par rapport à une stratégie gourmande (au plus court). Notre stratégie de pondération des liens est la suivante. Leur coût débute à 1, le premier flux alloué suivra donc son plus court chemin. A chaque nouvelle allocation, le poids de l'arc est augmenté en fonction du poids du flux alloué, selon la formule suivante :

$$coût_L = 1 + (Ch_L - R_L) \cdot agressivité \tag{2.30}$$

La valeur d'agressivité permet, comme son nom l'indique, de déterminer l'agressivité de la répartition de charge. Plus elle est élevée, plus les coûts augmenteront rapidement, obligeant les prochains flux à effectuer des détours. Plus elle est faible, plus la répartition de charge sera effectuée tardivement. Si les flux de communication sont répartis de façon relativement équilibrée en termes de nœuds source et destination, on peut procéder à une répartition de charge peu agressive ; cela évitera de proposer des détours inutiles aux flux. En revanche, si les flux sont concentrés, il est plus efficace d'augmenter l'agressivité, et ainsi le caractère préventif de la répartition de charge.

Deux exemples de répartition de charge par évolution du coût des liens sont donnés en Figure 2.14.

2.6.3 Algorithmes de recherche de chemin

Identification du problème

L'algorithme de recherche de chemin doit sélectionner le chemin le moins coûteux parmi ceux qui satisfont l'équation 2.29. Il s'agit d'un problème de plus court chemin restreint, ou *Restricted Shortest Path* (RSP). Ce problème est un cas particulier du problème de recherche de chemin optimal multi-contraint (*Multi-Constrained Optimal Path*, MCOP), lorsqu'une seule fonction de coût est considérée, ce qui est le cas ici avec notre optimisation. Dans le cas où l'on souhaiterait

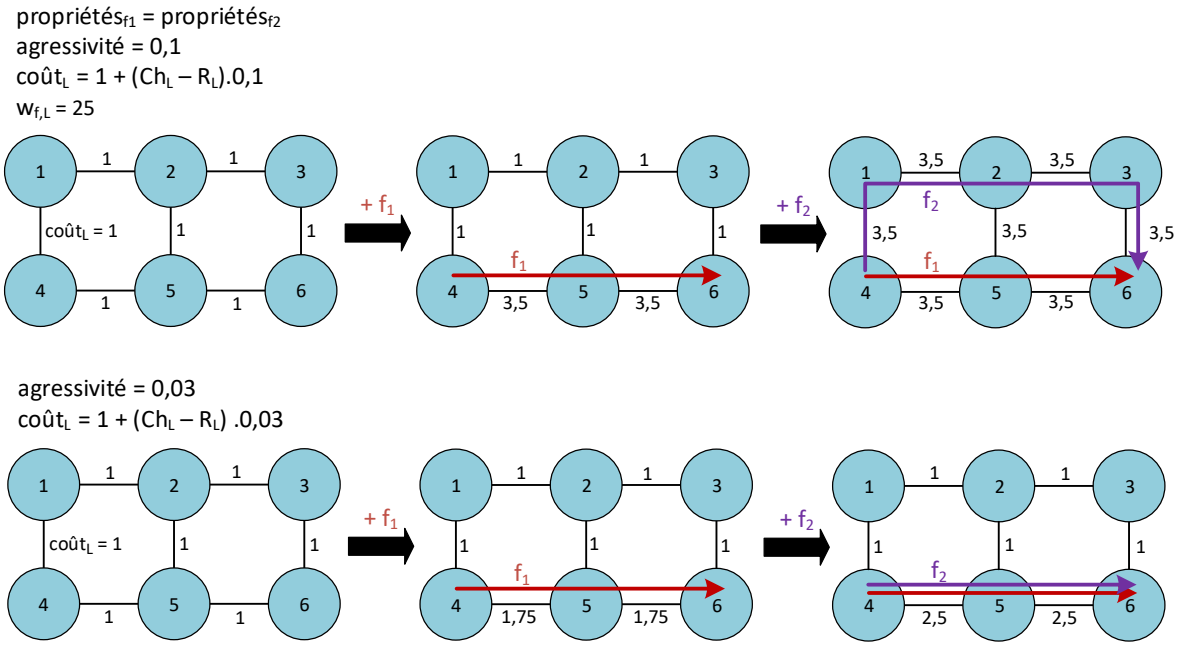

FIGURE 2.14 – Exemple de répartition de charge due à l'évolution du coût des liens

optimiser le chemin selon plusieurs critères, le problème deviendrait un problème de recherche de chemin optimal multi-contraint.

Description des algorithmes

Le problème de *Restricted Shortest Path* a été montré comme NP-complet [Ahuja et al., 1993], bien que [Van Mieghem and Kuipers, 2003] suggère que ce ne soit pas le cas en pratique, dans des réseaux réalistes. De nombreuses solutions, exactes ou optimistes, ont été proposées. On citera par exemple [Hassin, 1992] et [Ni et al., 2011]. [Korkmaz and Krunz, 2001] propose d'autres recherches et étend la revue au problème MCOP. L'objectif de notre étude n'étant pas de traiter du problème de recherche de chemin en profondeur, nous allons utiliser une heuristique simple et viable en temps réel. La modularité de l'architecture SDRN permet cependant de remplacer l'algorithme de recherche de chemin par une autre solution au problème de RSP si l'on souhaite optimiser ce point.

Nous proposons une version modifiée de l'algorithme de Yen [Yen, 1971]. Cet algorithme permet de calculer les k-plus courts chemins dans un graphe, en s'appuyant sur un algorithme de recherche de plus court chemin entre couple de sommets. Pour ce dernier, nous choisissons l'algorithme de Dijkstra [Dijkstra, 1959]. Deux versions de l'algorithme de Dijkstra seront utilisées dans nos algorithmes. La fonction *dijkstra_coût* retourne le chemin le moins coûteux entre la source et la destination, c'est-à-dire le chemin le plus optimisé. La fonction *dijkstra_sauts* retourne le chemin le plus court en termes de nombre de sauts (taille), sans prendre en compte les coûts des arcs. Ces fonctions sont utilisées dans les fonctions *yens_coût* et *yens_sauts*, qui calculent le $k^{ème}$ plus court chemin, en termes de coût et taille, respectivement.

On définit alors deux algorithmes :

— **méthode A** : chercher les chemins les moins coûteux (les plus optimisés), jusqu'à ce qu'un vérifie l'équation 2.29 (Algorithme 4) ;

— **méthode B** : chercher les chemins vérifiant l'équation 2.29, puis sélectionner le moins coûteux (Algorithme 5).

On inclut à ces algorithmes un paramètre temporel, `stopTime`. Il permet de contrôler le temps d'exécution de l'algorithme. Si ce temps est atteint, alors l'algorithme retourne une solution rapide, potentiellement sous-optimale.

Enfin, si aucune solution ne permet de respecter la contrainte de latence, c'est-à-dire qu'il n'existe aucun chemin suffisamment court entre la source et la destination, alors on augmente le poids du flux au-delà du minimum requis, de sorte à réduire le délai pire cas. Si ce poids dépasse le nombre total de sous-canaux Ch_L, alors il n'existe aucune solution. La fonction d'*initialisation* (Algorithme 3) calcule les paramètres protocolaires, réduit le graphe en conséquence, calcule la taille maximale de chemin, puis vérifie qu'au moins une solution existe. Si une solution existe, elle retourne les paramètres protocolaires éventuellement altérés, le graphe réduit et la taille maximale de chemin. Sinon, on peut terminer directement le contrôle d'admission : le flux est refusé.

Algorithme 3 : Initialisation

```
1  Fonction initialisation(G_R, requête_f, propriétés_R)
2      w_{f,LI} = équation 2.18(requête_f, propriétés_R) ;
3      w_{f,LE} = équation 2.18(requête_f, propriétés_R) ;
4      plus_court_chemin : Chemin ;
5      tant que vrai faire
           /* Réduction du graphe                                   */
6          G_R_temp ← clone(G_R) ;
7          pour L ∈ G_R_temp faire
8              si R_L < w_{f,L} alors
9                  G_R.retirer(L) ;
10             fin
11         fin
           /* Calcul de la taille maximale de chemin                */
12         taille_maximale_f ← équation 2.29(requête_f, propriétés_R) ;
           /* Existe-t-il une solution ?                            */
13         plus_court_chemin ← dijkstra_coût(G_R_temp, requête_f.src_f, requête_f.dest_f) ;
14         si taille(plus_court_chemin) > taille_max_f alors
               /* Aucune solution                                   */
15             w_{f,LI}++ ;
16             w_{f,LE}++ ;
17             si w_{f,L} > Ch_L alors
18                 retourner null ;
19             fin
20         fin
21         sinon
22             G_R ← G_R_temp ;
23             retourner {G_R_temp, taille_max_f, w_{f,LI}, w_{f,LE}} ;
24         fin
25     fin
```

Algorithme 4 : Méthode A

```
1  Fonction méthodeA(G_R, requête_f, propriétés_R, stopTime)
2      {G_R_temp, taille_max_f, w_f,LI, w_f,LE} ← initialisation(G_R, requête_f, propriétés_R) ;
       /* On recherche le k^ème moins coûteux jusqu'à ce qu'il soit suffisamment
          court ou que le temps soit écoulé                                    */
3      k ← 0 ;
4      tant que vrai faire
5          k++ ;
6          chemin ← yens_coût(k, G_R, requête_f.src_f, requête_f.dest_f) ;
7          si taille(chemin) ≤ taille_max_f alors
8              retourner chemin ;
9          fin
10         sinon si temps_écoulé > stopTime alors
11             retourner plus_court_chemin ;
12         fin
13         sinon
14             continuer ;
15         fin
16     fin
```

L'évaluation de ces algorithmes est effectuée en Section 4. Cependant, une première intuition nous permet de concevoir une troisième méthode de recherche de chemin. En effet, la méthode A termine dès qu'elle a trouvé un chemin respectant la contrainte de taille. Dans le meilleur cas, elle peut donc terminer très rapidement, dès la première itération. Dans un cas moins idéal, elle peut commencer par un chemin très long (mais peu coûteux), puis boucler très longtemps avant de trouver un chemin qui satisfasse l'équation 2.29. La méthode B est plus stable. Son temps d'exécution minimum est plus long, car elle calculera toujours un ensemble de chemins avant de sélectionner le plus optimal. En revanche, elle se déroule toujours en partant du plus court chemin. Son temps d'exécution maximum dépend de la taille maximale du chemin autorisée, mais il ne devrait pas risquer d'atteindre des niveaux aussi excessifs que celui de la méthode A. En résumé, l'intuition est que la méthode A permet des temps d'exécution très courts, mais n'est pas intéressante à partir d'un certain seuil de temps d'exécution. Cette intuition nous conduit à penser un algorithme hybride, profitant des avantages des deux méthodes : temps d'exécution minimum de la méthode A et stabilité du temps d'exécution maximum de la méthode B. Cet algorithme, nommé **méthode hybride X/Y**, consiste à exécuter la méthode A pendant X% du temps alloué, puis la méthode B pendant le reste du temps, soit Y% du temps alloué. On a donc $X + Y = 100$. Cette méthode est résumée dans l'Algorithme 6.

Évaluation

Soit m le nombre d'arcs dans le graphe. La complexité de l'algorithme de Dijkstra peut être réduite à $O(m + n.log(n))$ avec la meilleure implémentation [Barbehenn, 1998]. La complexité de la recherche des k-plus courts chemins avec l'algorithme de Yen est de $O(k.n(m + n.log(n)))$ dans le pire cas. Elle correspond à $k.n$ appels à l'algorithme de Dijkstra.

On évalue ensuite les trois algorithmes présentés ci-dessus : méthode A, méthode B et méthode hybride X/Y. L'évaluation est réalisée via une implémentation en Java 1.8 du contrôleur, sur des

Algorithme 5 : Méthode B

```
1  Fonction méthodeB(G_R, requête_f, propriétés_R, stopTime)
2      {G_R_temp, taille_max_f, w_f,LI, w_f,LE} ← initialisation(G_R, requête_f, propriétés_R) ;
       /* On recherche le k^ème plus court chemin, jusqu'à ce qu'un soit trop
          long ou que le temps soit écoulé                                    */
3      chemins ← Liste<Chemin> ;
4      k ← 0 ;
5      tant que temps_écoulé < stopTime faire
6          k++ ;
7          chemin ← yens_sauts(k, G_R, requête_f.src_f, requête_f.dest_f) ;
8          si taille(chemin) > taille_max_f alors
9              casser la boucle ;
10         fin
11         chemins.ajouter(chemin) ;
12     fin
       /* On retourne le chemin le moins coûteux parmi ceux calculés          */
13     résultat ← chemins.accéder(1) ;
14     pour chemin dans chemins faire
15         si coût(chemin) < coût(résultat) alors
16             résultat ← chemin ;
17         fin
18     fin
19     retourner résultat ;
```

Algorithme 6 : Méthode hybride X/Y

```
1  Fonction méthodeHybride(G_R, requête_f, propriétés_R, stopTime, X)
2      {G_R_temp, taille_max_f, w_f,L_i, w_f,L_e} ← initialisation(G_R, requête_f, propriétés_R) ;
3      méthodeA(G_R_temp, requête_f, propriétés_R, stopTime × X/100) ;
4      méthodeB(G_R_temp, requête_f, propriétés_R, stopTime × (100 − X)/100) ;
```

TABLE 2.4 – Évaluation des temps d'exécution moyens et pourcentage d'occurences atteignant le temps alloué

Algorithme	Nombre de nœuds	Temps moyen	Dépassement du temps alloué
Méthode A	25	0,68 ms	0,93 %
Méthode B	25	1,76 ms	0,01 %
Hybride 25-75	25	1,04 ms	0,03 %
Hybride 75-25	25	1,06 ms	0,49 %
Méthode A	100	23,22 ms	10,65 %
Méthode B	100	36,53 ms	0,01 %
Hybride 25-75	100	22,14 ms	5,06 %
Hybride 75-25	100	22,60 ms	5,19 %

ordinateurs Raspberry Pi 3 modèle B, possédant un processeur ARMv8 64-bits quad-core à 1.2 GHz et 1 Go de mémoire vive. Le réseau est modélisé par une matrice d'adjacence. Nos trois critères d'évaluation sont : la distribution du temps d'exécution, sa moyenne et le nombre d'occurences atteignant le temps alloué. Ce dernier représente le fait que l'algorithme s'est exécuté pendant trop longtemps et a dû être stoppé, retournant une solution rapide potentiellement sous-optimale.

Le protocole d'évaluation est le suivant :

1. Générer un réseau tore symétrique de $taille \times taille$ nœuds. Dans notre cas, 25 puis 100 nœuds ;
2. Générer 250 requêtes de flux aléatoires (dans des intervalles réalistes), telles qu'entre 5% à 10% d'entre elles n'ont pas de solution. Le temps alloué est de 10 ms pour le réseau de 25 nœuds et 100 ms pour le réseau de 100 nœuds. Ces valeurs de seuil ont été choisies de sorte que le nombre d'occurences les atteignant soit suffisamment grand pour être perceptible, mais le plus petit possible pour ne pas biaiser l'évaluation du temps d'exécution moyen ;
3. Calculer le temps d'exécution moyen du traitement d'une requête de flux ;
4. Vider le réseau et recommencer les étapes 2 à 4 sur un total de 1 000 itérations.

Les résultats de cette évaluation sont donnés en Tableau 2.4, Figure 2.15 et Figure 2.16. Les résultats ont été discrétisés afin de représenter la tendance de la courbe des temps d'exécution. Nous avons préféré relier les points plutôt que les présenter sous forme d'un histogramme, pour des raisons de lisibilité de la tendance. Pour cette raison, on observe une pente croissante en fin de courbe au lieu d'une barre verticale. Notons que les distributions des temps d'exécutions pour les deux versions de la méthode hybride sont similaires, seuls les taux de dépassement du temps alloué sont significativement différents (cf. Tableau 2.4).

Premièrement, on remarque que la version 25/75 de la méthode hybride est toujours meilleure que la version 75/25. Cela signifie que si l'algorithme n'a pas trouvé de solution durant le premier quart du temps alloué, il bouclera généralement encore longtemps avant de la trouver. Cela met en avant l'intérêt d'utiliser la méthode A pendant une partie réduite du temps alloué. Notons que le ratio d'hybridation optimal n'est pas étudié dans ce document.

Deuxièmement, la méthode hybride 25/75 montre des résultats très intéressants dans le cas du réseau de 25 nœuds. Elle propose le meilleur compromis entre optimalité des solutions (temps alloué peu atteint) et temps d'exécution moyen. Dans le cas du réseau de 100 nœuds, l'intérêt

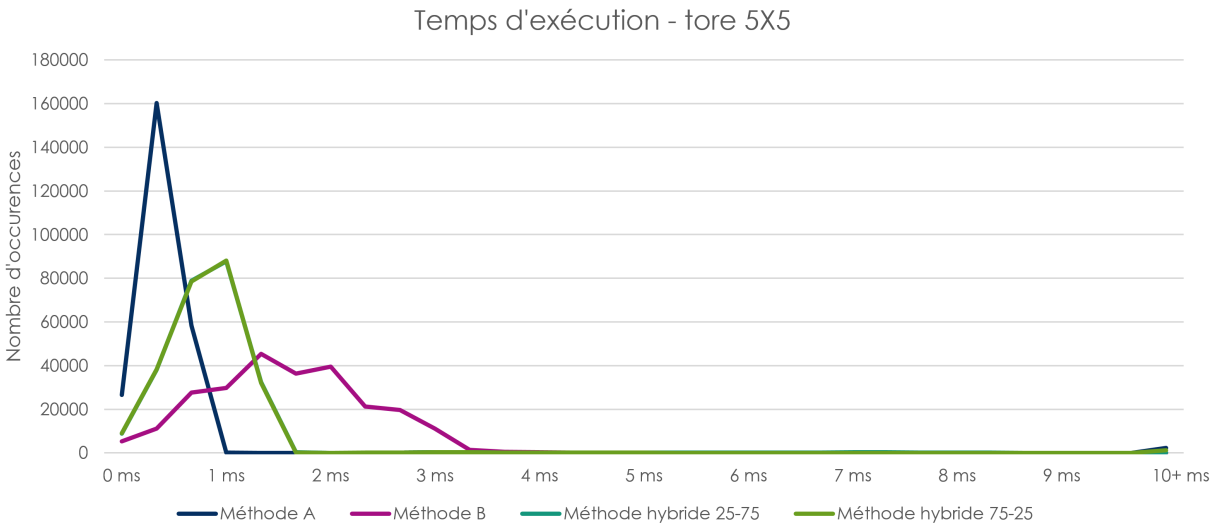

FIGURE 2.15 – Distribution des temps d'exécution, réseau tore 5X5

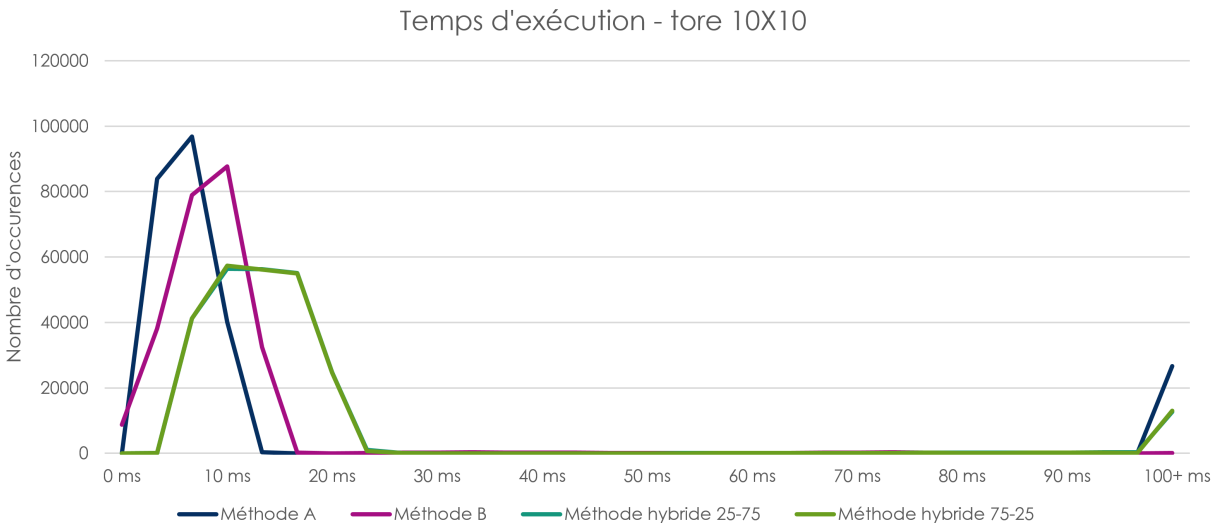

FIGURE 2.16 – Distribution des temps d'exécution, réseau tore 10X10

de la méthode hybride est moins marqué. Bien qu'elle propose un meilleur compromis que la méthode A, très instable dans ce scénario, elle ne réduit le taux de dépassement du temps alloué que de moitié. Cela est dû au fait que, dans un si grand réseau, une seule boucle de la méthode A peut consommer tout le temps alloué. Or, le passage d'une méthode à l'autre est effectué en fin de boucle. Si l'interruption pouvait avoir lieu à n'importe quelle étape de la méthode A, il est probable que la méthode hybride proposerait des résultats similaires à la première expérience. Notons enfin qu'un réseau de 100 nœuds est un scénario limite compte tenu des cas réels dans l'industrie embarquée. Il nous permet de montrer la faisabilité de notre approche dans les réseaux les plus grands.

Troisièmement, le dépassement du temps alloué n'a pas la même valeur en fonction de la méthode. En effet, si la méthode A atteint le temps alloué, elle retournera le plus court chemin, probablement sous-optimal. En revanche, si la méthode B atteint le temps alloué, elle retournera le chemin le plus optimisé parmi l'ensemble des chemins calculés jusqu'alors. La probabilité que le chemin retourné soit sous-optimal, ainsi que l'intensité de cette sous-optimalité, sont donc plus faibles. La résultats de la méthode B et, par extension, de la méthode hybride, sont donc beaucoup moins affectés par les dépassements du temps alloué. Ceci conforte les conclusions données ci-dessus et l'intérêt de la méthode hybride.

Enfin, une interruption immédiate de l'algorithme lors du dépassement du temps alloué permettrait de garantir aux applications une durée maximale de contrôle d'admission. Ceci n'est bien sûr possible qu'à condition que le temps alloué soit raisonnable, c'est-à-dire qu'il permette au moins le calcul d'un chemin.

2.6.4 Interface avec les applications

Dans un système communicant sur une architecture SDRN, le contrôleur est au service des applications. Celles-ci lui envoient des requêtes correspondant aux flux qu'elles souhaitent allouer sur le réseau. Le contrôleur est chargé de les analyser ainsi que l'ensemble des propriétés du réseau, pour en déduire, si elle existe, une configuration optimale du matériel garantissant le respect des contraintes temps réel des flux en toute circonstance.

Cependant, nous souhaitons que la couche applicative puisse décider de l'allocation effective ou non d'un flux, après avoir reçu la ou les réponses du contrôleur. Ceci permet de répondre à plusieurs scénarios :

- si l'application possède plusieurs flux critiques, elle doit s'assurer qu'ils peuvent tous être alloués avant de demander la configuration du matériel et démarrer la communication ;
- si la couche applicative souhaite tester plusieurs configurations de déploiement d'applications, elle va faire appel plusieurs fois au contrôleur pour obtenir des informations sur ce qu'il est possible d'allouer, puis prendre une décision ;
- si un chemin critique applicatif est constitué d'une chaîne de flux de communication dont le délai bout-en-bout est contraint, la couche applicative doit traiter les flux concernés dans leur ensemble.

Une application (ou couche de gestion des applications) peut donc accepter ou décliner l'allocation d'un flux proposée par le contrôleur. Elle peut et doit également les libérer lorsqu'elle n'en a plus l'utilité. Pour ce faire, elle utilise les fonctions `accepter(`id_f`)`, `décliner(`id_f`)` et `libérer(`id_f`)` du contrôleur. id_f est un identifiant du flux, unique dans le réseau et défini par le contrôleur. Si un flux est accepté par le contrôleur, celui-ci retourne l'identifiant de flux correspondant à la couche applicative et pré-alloue le flux dans son modèle. Sinon, il retourne l'identifiant 0. Lorsque la couche applicative accepte ou libère un flux, le contrôleur configure le matériel selon les paramètres sauvegardés dans le modèle. Lorsqu'elle décline un flux, celui-ci

est retiré du modèle sans interaction aucune avec le matériel. Il est important de noter que la pré-réservation d'un flux dans le modèle ne doit pas impacter les autres contrôles d'admission du même flux. En effet, une seule instance sera, *in fine*, acceptée et allouée. Le contrôleur doit donc retirer temporairement du modèle toutes les autres instances du même flux avant de réaliser le contrôle d'admission.

De plus, comme discuté en Section 2.6.3, nous souhaitons donner la possibilité à la couche applicative de définir un temps maximum de contrôle d'admission. En fonction de l'urgence de l'allocation, l'application peut ainsi régler le niveau d'optimalité du chemin calculé. Si ce temps n'est pas raisonnable, c'est-à-dire qu'il ne permet pas le calcul d'au moins un chemin, alors il ne sera pas pris en compte. Toujours dans une optique de personnalisation des requêtes, on ajoute à celles-ci un paramètre lié à la stratégie d'optimisation. Il est optionnel mais peut servir à fournir des informations supplémentaires au contrôleur concernant la stratégie d'optimisation. Par exemple, dans une stratégie de regroupement des flux de même types de données, ce paramètre peut servir à indiquer les flux liés. On obtient donc : $requête_f = \{propriétés_f, échéance, optim\}$.

Les interactions entre les applications et le contrôleur sont représentées dans les diagrammes de séquence donnés en Figure 2.17.

2.6.5 Interface avec le matériel

Lorsqu'un flux a été accepté par le contrôleur puis par la couche applicative, ou libéré par la couche applicative, le contrôleur est responsable de la configuration du matériel. Le processus de configuration est défini par le module matériel concret, cependant il utilisera toujours les trois informations suivantes :

— l'identifiant de flux, id_f ;
— les paramètres protocolaires du flux, $\rho_{f,\mathcal{R}}$;
— le chemin défini pour le flux, $path_f$.

Ces trois paramètres sont regroupés dans un ensemble, $conf_f = \{id_f, \rho_{f,\mathcal{R}}, path_f\}$.

Il est important que l'identifiant de flux ne soit pas utilisé directement par les applications lors des communications, car on souhaite conserver la transparence de l'architecture SDRN.

Les interactions de configuration entre le contrôleur et le matériel sont représentées dans les diagrammes de séquence donnés en Figure 2.17.

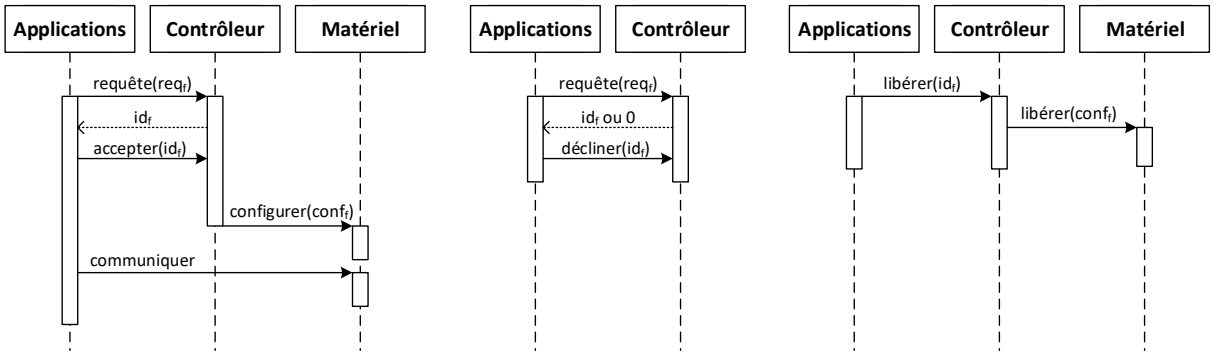

FIGURE 2.17 – Diagrammes de séquence des interfaces contrôleur/applications et contrôleur/matériel

2.6.6 Module abstrait

Le fonctionnement du contrôleur décrit dans cette partie peut être généralisé afin d'en déduire le module abstrait correspondant. Les interfaces sont conservées mais plusieurs éléments internes peuvent être remplacés :

- La méthode de recherche de chemin (algorithme de Yen) peut être remplacée par un autre algorithme de recherche de plus court chemin restreint (RSP). Si la stratégie d'optimisation utilise plusieurs critères, le problème devient un problème de recherche de chemin optimal multi-contraint (MCOP). Des solutions telles que celles revues dans [Ben Slimane et al., 2015] doivent alors être utilisées ;
- La stratégie d'optimisation peut être différente de la stratégie de répartition de charge choisie dans notre solution concrète ;
- Le réseau peut être modélisé de différentes façons.

Le module contrôleur abstrait contient donc un algorithme de RSP ou MCOP, une stratégie d'optimisation et un modèle du réseau, qui doivent être concrétisés par le concepteur du système (Figure 2.18).

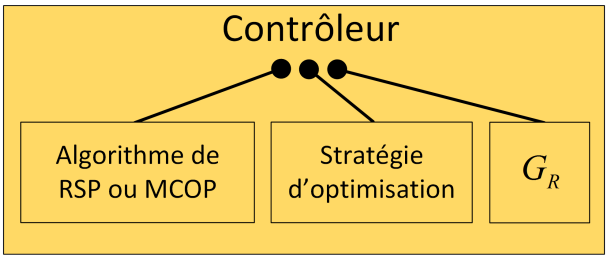

FIGURE 2.18 – Module contrôleur abstrait

On déduit également un algorithme de contrôle d'admission haut niveau, qui doit être précisé en fonction des éléments précédents (Algorithme 7).

Algorithme 7 : Résumé haut niveau du contrôle d'admission

```
1 Fonction contrôle_admission(G_R, requête_f, propriétés_R)
2     calculer les paramètres protocolaires ;
3     calculer la contrainte sur la taille du chemin ;
4     retirer du modèle les liens ne pouvant fournir les paramètres protocolaires ;
5     appliquer une méthode de RSP ou MCOP pour chercher le chemin optimal parmi
        ceux respectant la contrainte de taille de chemin ;
6     répondre à la couche applicative en fonction du résultat ;
```

2.7 Architecture SDRN globale détaillée

En synthèse l'architecture SDRN est composée de plusieurs modules abstraits, interagissant entre eux par des interfaces. En pratique, les modules doivent être concrétisés en respectant les interfaces.

La couche applicative envoie des requêtes $requête_f = \{propriétés_f, échéance, optim\}$ au contrôleur. Ce dernier utilise sa connaissance de l'ordonnancement (fonctions protocolaire, de

TABLE 2.5 – Caractéristiques des flux

	T_f	S_f	S_f/T_f	dl_req_f	src_f	$dest_f$
f_1	16,7 ms	2,868 Mbit	172 Mbit/s	50 ms	N_8	N_6
f_2	100 ms	960 bit	9,6 kbit/s	10 ms	N_5	N_8
f_3	5 ms	128 bit	25,6 kbit/s	25 ms	N_5	N_2
f_4	10 ms	5 kbit	500 kbit/s	10 ms	N_7	N_2

délai de saut et de délai de rafale) et des propriétés du réseau bas niveau pour traiter la requête de flux. Il cherche une configuration faisable et optimisée pour le flux, du type $conf_f = \{id_f, \rho_{f,\mathcal{R}}, path_f\}$: identifiant, paramètres protocolaires, chemin. Si une telle configuration existe, il répond favorablement à la couche applicative. En fonction des décisions d'allocation ou de libération des flux, le contrôle configure le matériel en cohérence avec les configurations préalablement calculées. Cette architecture est synthétisée en Figure 2.19.

Dans ce chapitre, nous avons proposé une solution concrète de l'architecture SDRN. Elle utilise notamment les éléments suivants : ordonnancement *Credit-Bounded Weighted Round Robin*, recherche de chemin avec l'algorithme de Yen, répartition de charge. Elle est synthétisée en Figure 2.20.

Au final, le modèle complet d'un flux, après contrôle d'admission, est le suivant :
$f = \{type_f,\ T_f,\ S_f,\ dl\_req_f,\ src_f,\ dest_f,\ id_f,\ \rho_{f,\mathcal{R}},\ path_f\}$, soit les caractéristiques du flux (période minimale, échéance, etc.) et les résultats du contrôle d'admission (identifiant, paramètres protocolaires, chemin).

2.8 Application numérique : système drone sur réseau RapidIO

Nous prenons pour exemple simple celui qu'un drone composé de quatre flux temps réel sur son réseau SDRN :

— f_1 : caméra infrarouge, produisant 60 images de 640×280×16 bits chaque seconde ;
— f_2 : contrôle caméra ;
— f_3 : centrale inertielle à une fréquence de 200 Hz ;
— f_4 : transit des communications externes.

Les caractéristiques des flux sont données dans le Tableau 2.5.

Le réseau est un tore RapidIO composé de 9 nœuds (Figure 2.21) et de liens identiques ($L = LE = LI$), avec $C_L = 1\,\text{Gbit/s}$, $Ch_L = 100$, $d_{sw,\mathcal{R}} = 0{,}01\,\text{ms}$, $\eta_L = 192\,\text{bit}$ et $\psi_L = 2048\,\text{bit}$.

Le choix du nombre de sous-canaux, Ch_L, correspond au choix d'un compromis entre la granularité et les contraintes d'implémentation. En effet, il est toujours plus intéressant, en termes d'allocation de ressources, de disposer de liens divisés en sous-canaux plus nombreux et plus petits. Cependant, cette division doit pouvoir être implémentée. Dans cet exemple, nous prenons un nombre de sous-canaux égal à 100, puisque cela correspond au nombre que nous avons choisi pour notre plateforme matérielle (voir Chapitre 6).

Le choix du quantum Q_L est lié aux contraintes d'échéance des flux imposants et des flux légers. Plus la valeur est haute, plus le délai pire cas sera :

— Faible pour les flux imposants, car les cycles sont plus longs, ce qui réduit le surcoût dû à la fragmentation ainsi que les pertes de crédit ;
— Élevé pour les flux légers, car le temps d'attente pire cas du jeton est plus long, étant donné qu'il accorde plus de temps par cycle à chaque sous-canal.

Pour cet exemple, nous utilisons trois valeurs de quantum :

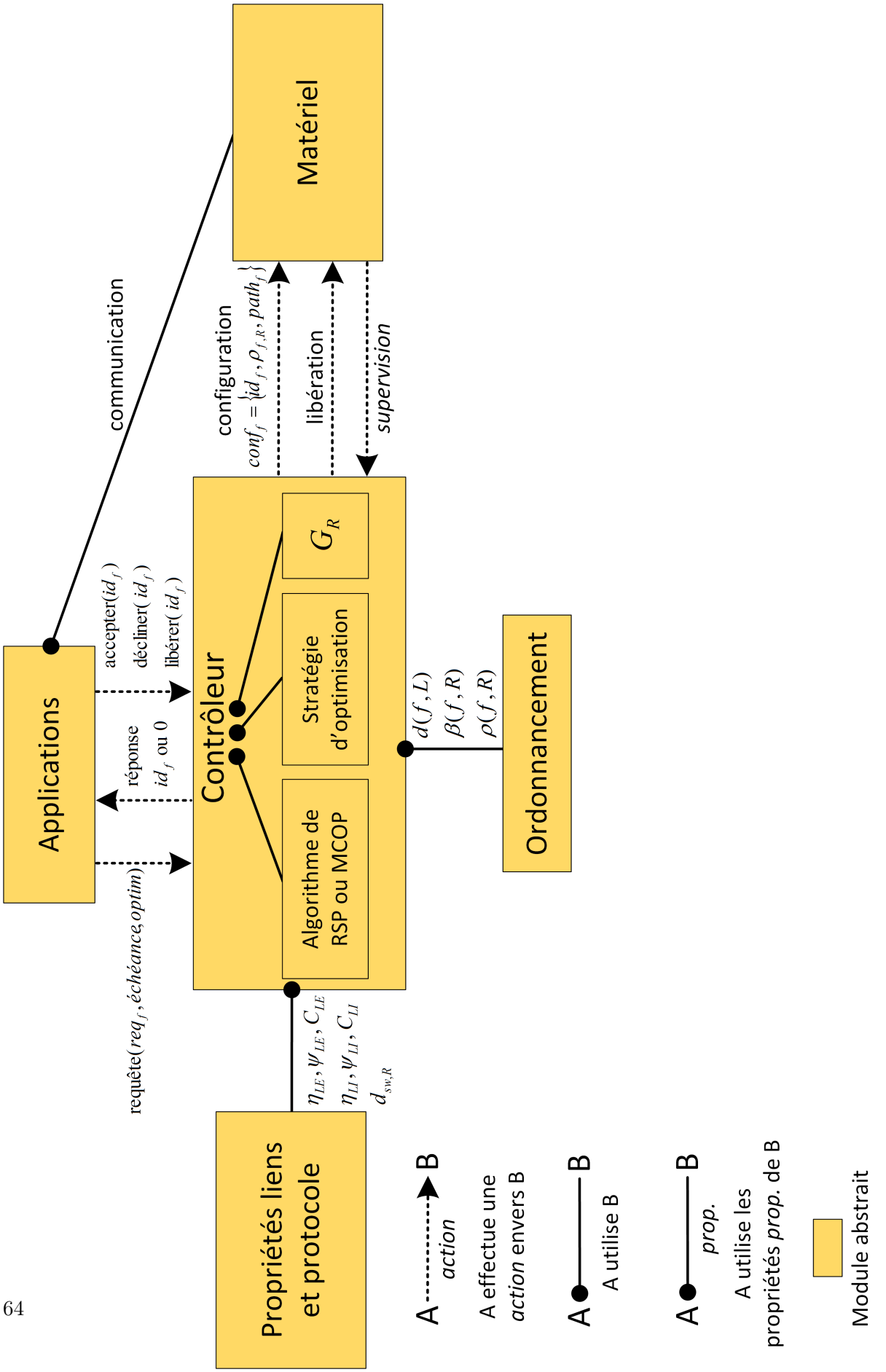

FIGURE 2.19 – Architecture SDRN abstraite détaillée

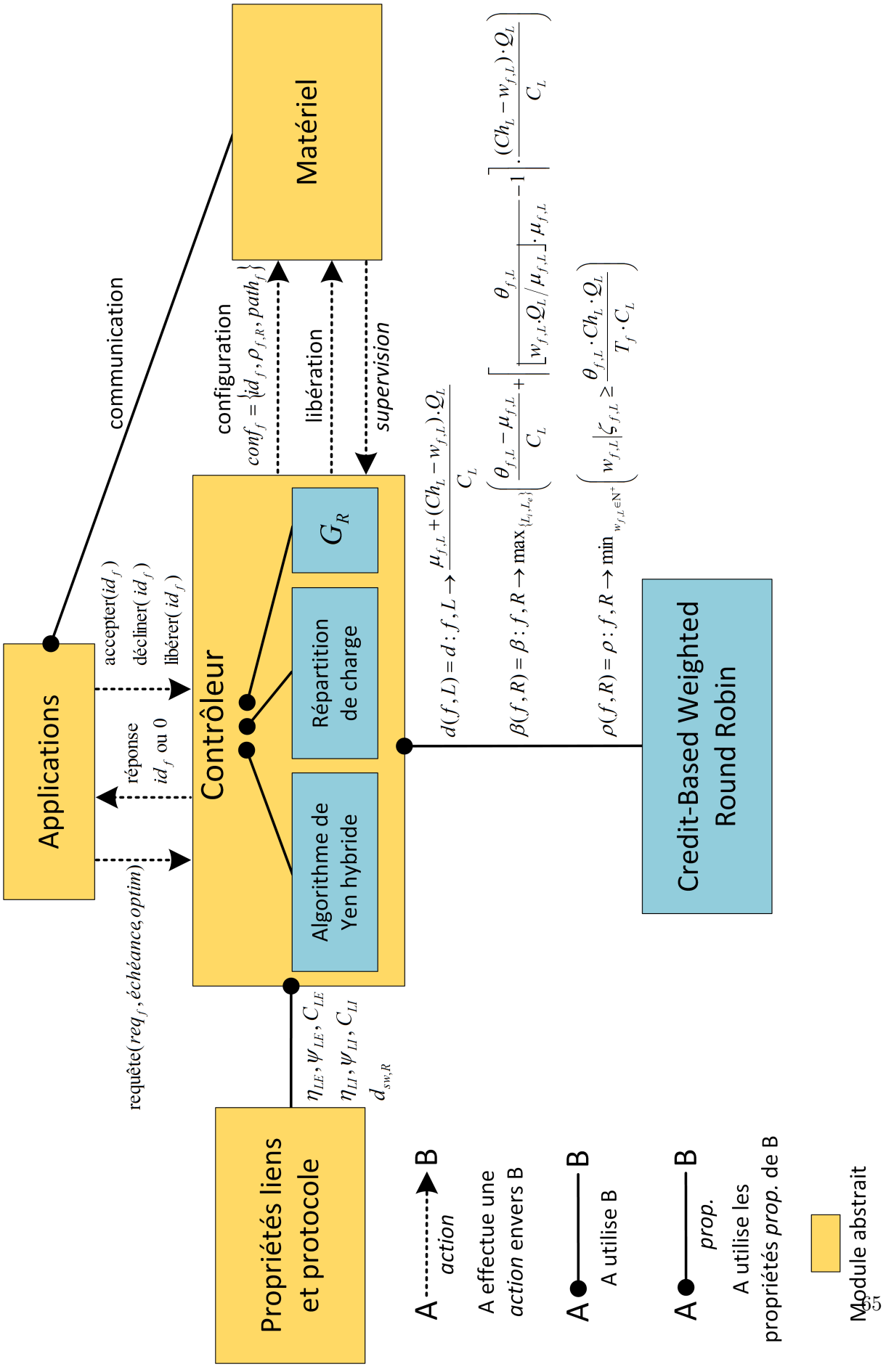

FIGURE 2.20 – Solutions concrète d'ordonnancement et de contrôle d'admission

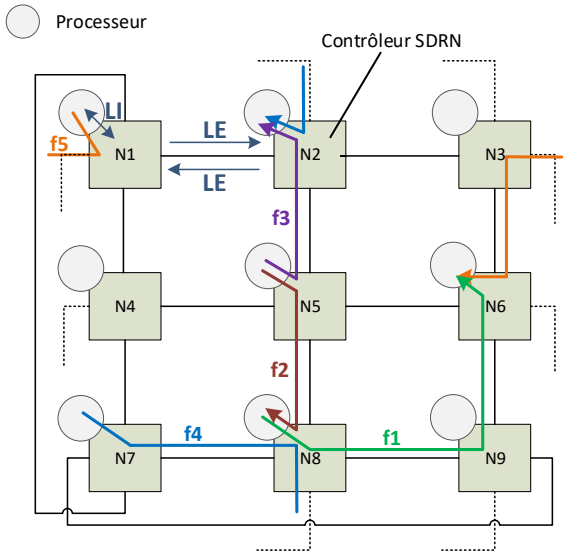

FIGURE 2.21 – Réseau tore RapidIO

— $Q_L1 = 500\,\text{bit}$;
— $Q_L2 = 2240\,\text{bit}$ (un paquet RapidIO de taille maximale) ;
— $Q_L3 = 8960\,\text{bit}$ (quatre paquets RapidIO de taille maximale).

On applique ensuite de façon incrémentale l'algorithme de contrôle d'admission pour les quatre flux :

1. Calculer le poids minimum requis via l'Algorithme 2 ;
2. Calculer la taille de chemin maximale via l'équation 2.29 ;
3. Trouver le chemin le plus optimisé via la méthode hybride 25/75 (Algorithme 6).

Les résultats de ce processus sont donnés en Tableau 2.6.

Comme cela peut être observé, une valeur plus importante de quantum Q_L implique un délai pire cas plus élevé pour les flux, y compris le plus imposant, f_1. Plus le flux est léger, plus la différence est observable. Ce système exemple bénéficierait donc d'une valeur de quantum plutôt faible, malgré le fait que cela implique une bande passante requise plus élevée pour f_1, dû à l'augmentation du surcoût.

2.9 Positionnement par rapport au paradigme SDN

Cette section positionne le fonctionnement de la configuration dynamique SDRN par rapport au paradigme *Software-Defined Networking* (SDN) usuel, tel que défini dans [Kreutz et al., 2015]. Une description plus complète des approches SDN est donnée en Chapitre 1.

On rappelle que les réseaux définis par logiciel, ou SDN, sont typiquement composés de trois plans (Figure 2.22) :

— Le plan de données : il s'agit du réseau physique ;
— Le plan de contrôle : un ou plusieurs contrôleurs capables de configurer le réseau physique par l'intermédiaire d'une interface dite *southbound*. L'interface la plus connue et utilisée est le protocole OpenFlow, standardisé par l'Open Networking Foundation [McKeown et al., 2008] ;

TABLE 2.6 – Résultats du contrôle d'admission

	Q_L	$w_{f,L}$	Taille de chemin max.	Délai pire cas	$path_f$
f_1	Q_L1	23	710	14,11 ms	{8, 9, 6}
f_2	Q_L1	1	163	0,32 ms	{5, 8}
f_3	Q_L1	1	417	0,17 ms	{5, 2}
f_4	Q_L1	1	152	1,03 ms	{7, 8, 2}
f_1	Q_L2	19	172	17,14 ms	{8, 9, 6}
f_2	Q_L2	1	41	0,69 ms	{5, 8}
f_3	Q_L2	1	106	0,69 ms	{5, 2}
f_4	Q_L2	1	39	1,37 ms	{7, 8, 2}
f_1	Q_L3	19	44	19,14 ms	{8, 9, 6}
f_2	Q_L3	1	10	2,68 ms	{5, 8}
f_3	Q_L3	1	26	2,68 ms	{5, 2}
f_4	Q_L3	1	10	3,59 ms	{7, 8, 2}

— Le plan de gestion ou applications réseau : ont une vision éventuellement simplifiée du réseau et prennent des décisions haut niveau sur la configuration. Elles communiquent avec le contrôleur par l'intermédiaire d'une interface dite *northbound*. Le contrôleur est ensuite chargé de faire la configuration effective. Il n'existe à l'heure actuelle pas d'interface *northbound* standard.

Les applications réseau ne sont pas à confondre avec les applications SDRN. Les premières gèrent le réseau, tandis que les secondes communiquent des données utiles à travers lui.

Un positionnement de l'architecture SDRN par rapport au paradigme SDN est donné en Figure 2.22. Le module matériel correspond au plan de données. L'interface contrôleur/matériel correspond à l'interface *southbound*. Cette interface étant découplée du contrôleur, il serait tout à fait possible d'utiliser une interface *southbound* SDN comme OpenFlow pour la configuration du matériel. Cependant, OpenFlow nécessite des composants compatibles avec le protocole, tandis que, dans la solution concrète que nous proposons, le protocole de configuration est transparent du point de vue du plan de données. Néanmoins, l'architecture SDRN est entièrement compatible avec une interface de configuration OpenFlow.

Dans SDRN, les plans de contrôle et de gestion ne sont pas séparés comme dans le paradigme SDN. Le contrôleur est chargé à la fois de la configuration effective du réseau et de la gestion plus haut niveau sur la base d'un modèle de flux.

La méthode de configuration effectuée dans SDRN est dite proactive. Cela signifie qu'on configure le matériel avant que les applications communiquent. Le paradigme SDN autorise une seconde forme de configuration, dite réactive. Le matériel n'est pas configuré *a priori*, il demande les informations au contrôleur lorsqu'il ne sait pas comment traiter un paquet. Cette méthode est intéressante pour des configurations complexes (réseaux très larges par exemple). Elle est cependant peu intéressante dans un contexte temps réel, puisqu'elle nécessiterait de garantir le temps de configuration par réaction.

Enfin, on peut noter une autre différence en la notion de flux. Les flux utilisés dans OpenFlow peuvent faire apparaître de nombreuses notions supplémentaires telles que les ports, les adresses IP et même des métacaractères. La notion est plus large et plus souple. Dans SDRN, les flux correspondent à des flux de communication précis, entre deux applications sur un sujet donné.

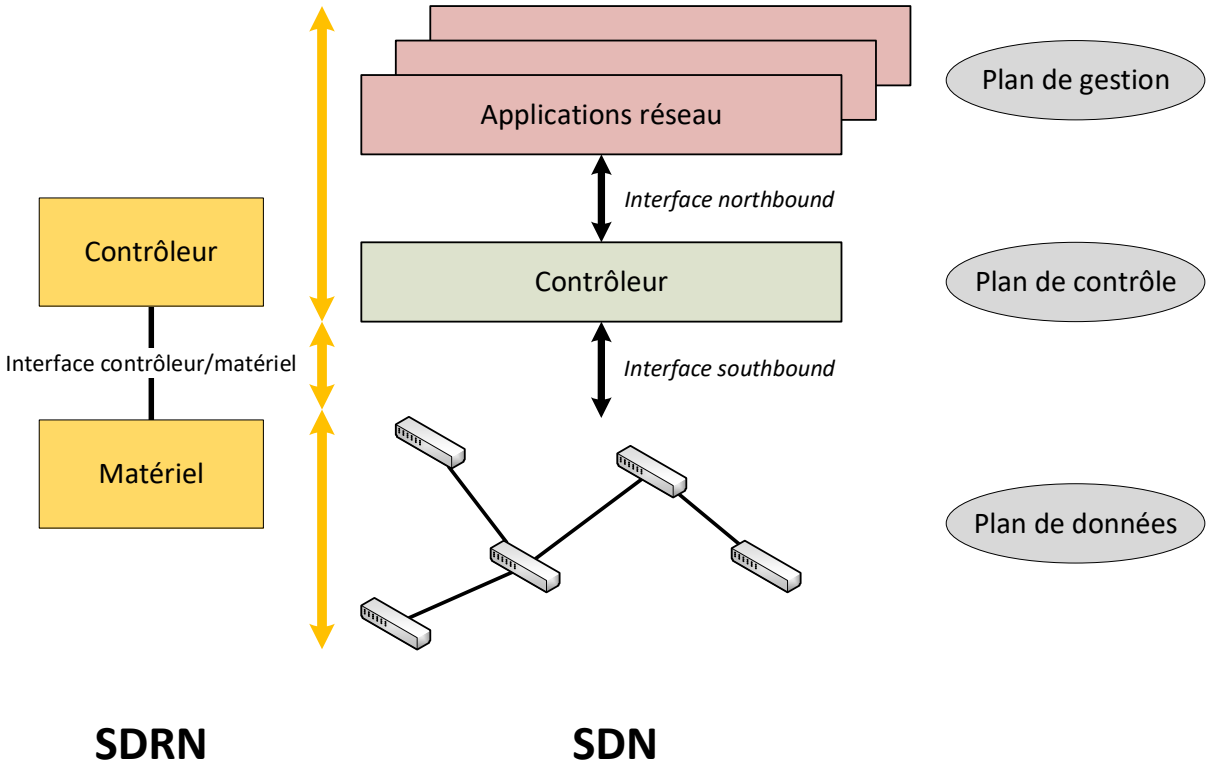

FIGURE 2.22 – Paradigmes SDN et SDRN

Ils sont uniquement liés à la couche applicative.

De manière générale, l'architecture SDRN est cohérente avec les grands principes du paradigme SDN usuel. C'est pour cela qu'on la définit également comme une architecture de réseau défini par logiciel.

2.10 Conclusion et perspectives

La configuration dynamique d'un réseau maillé embarqué nécessite de coupler plusieurs concepts qui ne sont pas étudiés ensemble dans la littérature : réseaux définis par logiciel, routage sous contraintes, analyse temporelle de flux temps réel, ordonnancement bas niveau, etc.

L'architecture SDRN présentée dans ce chapitre se positionne comme une solution bout-en-bout de réseau temps réel dynamique défini par logiciel. Elle spécifie comment les mécanismes et propriétés bas niveau du réseau peuvent être utilisés par un contrôleur pour configurer le matériel en fonction des besoins dynamiques des applications. Elle identifie ce qui est nécessaire à bas niveau et s'adapte aux propriétés du protocole, ce qui lui permet d'être applicable sur différents réseaux. Enfin, elle permet de valoriser les topologies résilientes telles que les topologies toriques ou maillées, grâce à une optimisation des chemins empruntés par les flux de communication. L'allocation dynamique des ressources est réalisée par un contrôleur SDRN présent sur un nœud du réseau. Les applications envoient des requêtes de flux au contrôleur. Ce dernier utilise ses connaissances du réseau pour en déduire si le flux peut être accepté, et si oui la configuration associée : identifiant, paramètres protocolaires, route. Selon les réponses du contrôleur, les applications confirment l'allocation des flux ou non, à la suite de quoi le matériel est configuré par le contrôleur. Un avantage majeur de SDRN est d'opérer une allocation des flux incrémentale et agnostique du trafic futur, ce qui signifie qu'il est possible de garantir le respect de contraintes temps réel à des flux de communication quelque soit l'évolution future du réseau. Cela signifie également que la complexité d'allocation est indépendante de la charge du réseau. En conséquence, le contrôle d'admission est utilisable en temps réel et passe à l'échelle.

La conception de l'architecture SDRN nécessite de répondre à de nombreuses problématiques scientifiques de natures différentes. Pour cette raison, elle a été conçue comme une architecture modulaire. Il est ainsi possible de remplacer un module par une autre version, sans remettre en cause ni maîtriser le reste de l'architecture. Par exemple, l'algorithme de recherche de chemin optimal peut être remplacé sans affecter l'architecture existante. Cinq modules, dits modules abstraits, composent l'architecture SDRN : applications, contrôleur, matériel, ordonnancement, propriétés liens et protocole. Ces modules doivent être implémentés sous la forme de modules concrets par le concepteur du système.

Dans ce chapitre, nous avons proposé une solution concrète bout-en-bout de réseau SDRN. Celle-ci utilise un ordonnancement de type *Credit-Bounded Weighted Round Robin*, une recherche de chemin basée sur l'algorithme de Yen et une stratégie de répartition de charge pour préparer l'avenir. Les liens sont divisés en sous-canaux virtuels alloués au flux. Cette méthode d'ordonnancement possède de bonnes propriétés d'isolation des flux et maximise la disponibilité de la bande passante lorsque celle-ci n'est pas utilisée par les flux critiques. Nous avons développé la méthode d'analyse temporelle utilisée par le contrôleur ainsi qu'une méthode heuristique de recherche de chemin optimal. Le contrôle d'admission a été évalué pour vérifier sa viabilité dans un contexte réel. Nous avons ensuite proposé un exemple numérique. Enfin, nous avons positionné l'architecture par rapport au paradigme SDN usuel, montrant sa cohérence et sa compatibilité avec les grands principes des réseaux définis par logiciel et leurs principaux standard.

Plusieurs perspectives d'évolution du SDRN peuvent être identifiées. Premièrement, notre

étude fait l'hypothèse qu'il n'y a pas de dysfonctionnement dans le réseau. Cette hypothèse sera relâchée dans les chapitres suivants.

Deuxièmement, les flux sont tous considérés comme *unicast*, c'est-à-dire ne communicant que vers une seule destination. Les communications *multicast* peuvent être gérées en les représentant par plusieurs flux. Cependant, cette méthode n'est pas très efficiente, car les paquets sont dupliqués sur les parties communes des chemins. Pour mutualiser ces parties communes et réaliser un véritable *multicast*, il est nécessaire d'adapter le modèle du réseau et la méthode de recherche de chemin. Cette dernière deviendrait une méthode de calcul d'arbres, c'est-à-dire de routes vers plusieurs destinations simultanées. Cependant, grâce à l'aspect modulaire du SDRN, cette évolution n'impacterait pas le reste de l'architecture.

Troisièmement, les flux sont représentés par un modèle périodique, notamment une période minimale entre deux envois de données. Si le flux n'est pas strictement périodique, cela n'est pas très grave car les ressources prévues en surplus sont utilisables par les communications non temps réel sans perte de bande passante. Cependant, une gestion dédiée des flux apériodiques permettrait au contrôleur de gérer un réseau dont le modèle est plus proche de la réalité. Dans certains cas, cela pourrait réduire le pessimisme et ainsi permettre à plus de flux d'être acceptés. Les flux apériodiques et légers, tels que des alertes, peuvent être fusionnés et rassemblées dans un même sous-canal virtuel. Il s'agit de flux critiques mais ne communicant que rarement voire jamais. Il est donc nécessaire de leur garantir une place dans le réseau, mais leur très faible charge permet de mutualiser les ressources qui leur sont allouées.

Quatrièmement, l'allocation incrémentale est associée à une préparation à l'avenir via la stratégie d'optimisation. Cela consiste, par exemple, à router un flux sur un chemin plus long si cela maximise les chances qu'un prochain flux critique à contrainte d'échéance très serrée soit accepté. Cependant, il serait possible d'améliorer cette préparation à l'avenir via une optimisation en tâche de fond par le contrôleur. Lorsqu'il ne traite pas de requête de flux, le contrôleur pourrait réorganiser le réseau si besoin. Autrement dit, modifier la façon dont les ressources sont allouées aux flux, de façon totalement transparente du point de vue des applications et sans remettre en cause les garanties de leurs contraintes temps réel.

Enfin, suite au contrôle d'admission, le contrôleur n'informe la couche applicative que de l'acceptation ou non du flux concerné. Il serait intéressant d'imaginer un retour plus personnalisé. Par exemple, indiquer à la couche applicative que le flux n'a pas été accepté car sa contrainte de délai ne peut être satisfaite. Cela permettrait à la couche applicative de s'adapter aux raisons du refus, dans cet exemple en rapprochant les applications source et destination.

En résumé, l'architecture SDRN présentée est une solution bout-en-bout modulaire à la problématique de réseaux maillés temps réel définis par logiciel. Elle identifie des points clé pour la mise en place d'un tel réseau. Elle spécifie également la façon dont un module peut être remplacé, afin, par exemple, de s'adapter à un autre protocole ou d'améliorer un algorithme précis. Enfin, elle est incrémentale, agnostique et asynchrone.

Chapitre 3

HYROS : routage hybride résilient pour la tolérance aux pannes

Sommaire

3.1 Introduction

Dans le Chapitre 2, nous avons présenté l'architecture SDRN. Celle-ci permet d'allouer des ressources réseau à des flux de communication temps réel, de façon dynamique et incrémentale.

Un contrôleur est chargé, à l'exécution, d'analyser les propriétés du réseau et des flux de communication pour en déduire une configuration réseau permettant de garantir au mieux le respect des contraintes temps réel. Le processus central à ce contrôleur est le contrôle d'admission. Il s'agit de chercher s'il existe une route permettant de respecter les contraintes temps réel du flux concerné, et si oui de définir la plus optimisée. Nous avons détaillé le processus de contrôle d'admission et proposé plusieurs algorithmes de recherche de route dans le réseau.

L'une de nos hypothèses de recherche (cf. Section 2.3) était la fiabilité complète du réseau, c'est-à-dire l'absence d'erreurs de transmission et de pannes. Dans cette partie, nous étudions le relâchement partiel de cette hypothèse. Plus précisément, nous étudions le processus de gestion des pannes de liens ou nœuds, autrement appelé **recouvrement de défaillance**.

L'un des avantages principaux du SDRN est la capacité à valoriser la résilience de la topologie réseau (par exemple, topologie maillée). Le contrôleur SDRN peut en effet profiter de la pluralité des chemins de communication et rapidement reconfigurer un flux affecté par une panne, de sorte qu'il la contourne ou suive un chemin complètement différent. De plus, grâce aux propriétés d'allocation incrémentale et d'isolation des modules contrôleur et d'ordonnancement (respectivement), cette reconfiguration peut être effectuée sans interférer avec les flux non concernés. Ainsi, la reprise après défaillance est effectuée par l'application du contrôle d'admission aux flux concernés, avec un modèle à jour du réseau. Durant cette période de reconfiguration, les propriétés temps réel des communications ne sont, par défaut, pas garanties. Des solutions telles que la duplication des flux sur des chemins disjoints permettent de garantir la présence d'au moins une route sûre durant la reconfiguration. Cependant, de telles méthodes sont coûteuses à la fois en termes de temps de calcul et de ressources réseau.

Dans cette partie, nous proposons un moyen efficient de gérer les périodes transitoires de reconfiguration. Notre contribution est basée sur la combinaison d'un routage à la source optimisé en période nominale avec un routage saut par saut en période de récupération de défaillance. Ce dernier apporte une flexibilité dans le contournement de chaque panne possible, mais est peu optimisé lors du passage à l'échelle. Pour cette raison, il n'est utilisé que lors des périodes transitoires. Une méthode permet de garantir à un flux temps réel qu'il continuera à respecter son échéance même pendant les périodes de reconfiguration, sans connaissance *a priori* sur le chemin de secours qui sera effectivement emprunté. Cette garantie de la résilience du chemin valorise les méthodes d'analyse temporelle appliquées par le contrôleur SDRN.

Après avoir rappelé quelques notions du SDRN, nous détaillerons le routage en mode nominal puis introduirons les problématiques liées à la gestion des périodes transitoires. Ensuite, nous décrirons le protocole de routage hybride résilient que nous proposons et comment il s'intègre à une architecture SDRN existante. Enfin, nous évaluerons sa faisabilité temps réel puis le positionnerons par rapport à d'autres approches, similaires ou non.

3.2 Rappels de notions SDRN

Dans cette section, nous rappelons quelques notions fondamentales du SDRN, nécessaires à la compréhension de la suite.

3.2.1 Architecture simplifiée

Le contrôleur SDRN est responsable, à l'exécution, de configurer le réseau de la façon la plus optimisée possible, considérant :

— Le modèle du réseau : topologie et trafic ;

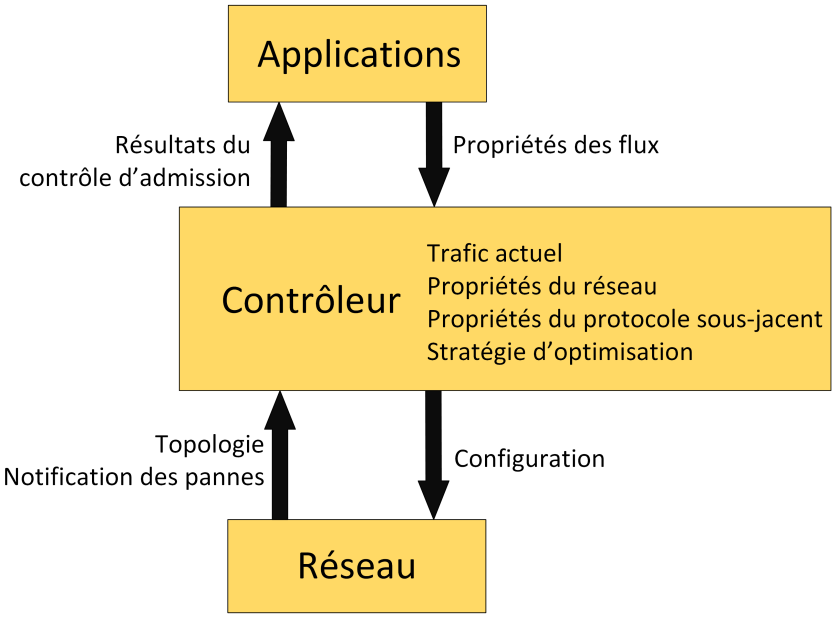

FIGURE 3.1 – Représentation simplifiée de l'architecture SDRN

— Les propriétés des liens et du protocole sous-jacent : bande passante, tailles des trames, fragmentation, etc. ;
— Les propriétés des flux : temps réel ou non, échéance relative, débit utile, etc. ;
— Une stratégie d'optimisation ; par exemple, une stratégie de répartition de charge.

L'architecture SDRN détaillée est l'objet du Chapitre 2. Nous en proposons une représentation simplifiée en Figure 3.1.

3.2.2 Division logique des liens

L'architecture SDRN s'appuie sur un module d'ordonnancement garantissant des propriétés d'isolation, déterminisme et équité minimale. Dans notre cas, nous avons choisi de diviser les liens en sous-canaux virtuels, auxquels est appliqué un ordonnancement de type *Credit-Bounded Weigthed Round Robin*. De plus, les liens sont divisés en deux grandes sous-parties, ou canaux :

— La partie temps réel utilisée pour l'allocation de flux temps réel ;
— La partie *best-effort*, utilisable librement mais sans aucune garantie individuelle. Cette partie utilise également un système de priorités pour apporter une notion de Qualité de Service. Un paquet n'est envoyé que s'il n'existe pas de paquet de niveau de priorité supérieur (valeur de priorité inférieure) dans la file d'attente de sortie.

La Figure 3.2 représente ce multiplexage.

3.2.3 Contrôle d'admission

Le contrôle d'admission est au cœur du mécanisme d'allocation des flux temps réel. Il est constitué de trois grandes parties (Figure 3.3) :

1. Analyse des contraintes : les requêtes de flux sont analysées en tenant compte des propriétés du réseau et protocole sous-jacent. Elles sont raffinées en deux contraintes, un poids minimal et une taille de chemin maximale. Cette étape sera essentielle à la mise en place de notre algorithme de routage ;

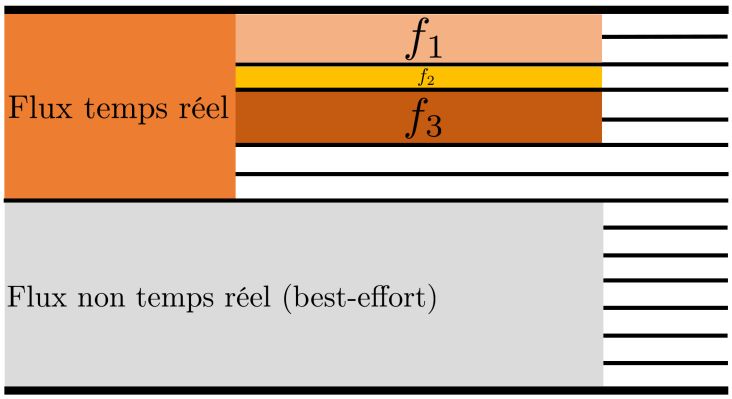

FIGURE 3.2 – Virtualisation des liens

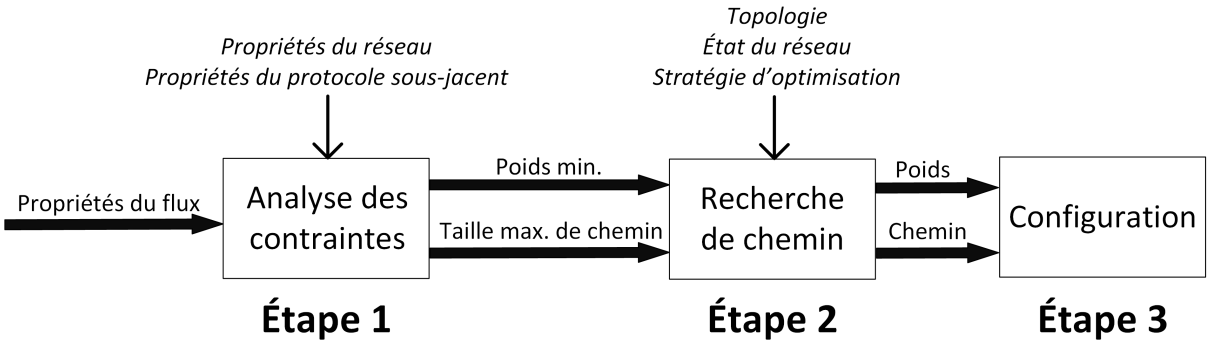

FIGURE 3.3 – Virtualisation des liens

2. Recherche de chemin : le chemin le plus optimal (s'il existe) est cherché, selon les deux contraintes énoncées ci-dessus ;
3. Configuration : si la recherche de chemin est positive et que l'application accepte l'allocation du flux, alors le matériel est configuré selon les paramètres retournés par la recherche de chemin : poids effectif et chemin.

3.3 Routage en mode nominal

3.3.1 Définitions

Dans cette sous-section, nous donnons quelques définitions relatives aux protocoles de routage.

Routage statique ou dynamique

Un protocole de routage statique, non adaptatif, ou *oblivious*, ne prend pas en compte l'état du réseau (par exemple, sa congestion) lorsqu'il prend une décision de routage. A l'inverse, un protocole de routage dynamique, ou adaptatif, adapte le chemin emprunté par les communications en fonction de l'état du réseau. Cette adaptation peut être réalisée lors de la première configuration des routes et/ou par reconfiguration lors de l'évolution de l'état du réseau. Un routage dynamique apporte une optimisation au prix d'une intelligence supplémentaire (surcoût).

TABLE 3.1 – Synthèse des caractéristiques des paradigmes de routage

	Statique ou dynamique	**Source ou saut par saut**	**Orienté machine ou donnée**
Surcoût	adaptatif > statique	source > saut par saut	donnée > machine
Optimisation	adaptatif > statique	source > saut par saut	donnée > machine
Adaptabilité	adaptatif = statique	saut par saut > source	donnée = machine

Routage à la source ou saut par saut

Dans un routage à la source, le chemin que le paquet va emprunter est connu avant son envoi. En général, l'émetteur définit la route puis l'embarque dans l'entête du paquet. Les routeurs ne prennent pas de décision sur la route et se contentent de suivre le chemin indiqué dans l'entête. D'autres implémentations que le stockage du chemin dans l'entête sont néanmoins envisageables. Par exemple, les routeurs peuvent être pré-configurés pour faire suivre au paquet le chemin défini à l'avance. L'idée fondamentale est que le chemin est défini à l'avance pour le paquet, les routeurs ne le construisent pas en fonction de sa destination. Dans un routage saut par saut, les routeurs définissent le chemin que le paquet doit emprunter, sur la base d'une information emportée par l'entête ; typiquement, l'adresse de la destination. Chaque routeur prend la décision sur le prochain voisin à visiter, sans connaissance du chemin déjà parcouru. Un routage à la source peut s'avérer plus cohérent et optimisé qu'un routage saut par saut, car il est effectué par la même entité avec une connaissance plus complète du réseau. Cependant, il n'est pas capable d'adapter la route au milieu du transport du paquet sur le réseau. Dans les réseaux embarqués, les flux temps réel suivent typiquement un routage à la source pour permettre la garantie du respect de leurs contraintes.

Routage orienté machine ou orienté donnée

Un routage orienté machine définit le chemin à emprunter en fonction de la machine sur laquelle se trouve le récepteur. Un routage orienté donnée, en revanche, définit le chemin à emprunter en fonction du contenu du paquet : type de donnée, flux associé, etc. Cela permet de transmettre un paquet sans besoin de connaître la destination, au prix d'une information supplémentaire sur la donnée ou le flux dont elle fait partie.

Synthèse

Le Tableau 3.1 synthétise les paradigmes de routage décrits précédemment, selon les caractéristiques qui nous intéressent dans notre étude :

- Surcoût de routage : intelligence supplémentaire nécessaire, se traduisant par un coût en temps de calcul ;
- Optimisation : utilisation maîtrisée et donc plus efficiente des ressources réseau ;
- Adaptabilité : capacité d'adapter la route en milieu de transmission.

3.3.2 Routage nominal dans les SDRN

Flux temps réel

En mode nominal, le chemin suivi par un flux temps réel est celui défini par le contrôleur lors du contrôle d'admission. Il s'agit donc d'un routage à la source, car l'information sur la route est connue avant l'émission du paquet. Cette information est distribuée sur les routeurs concernés sous la forme d'entrées de table de routage de format $id_f \rightarrow port$. Le routage est donc orienté

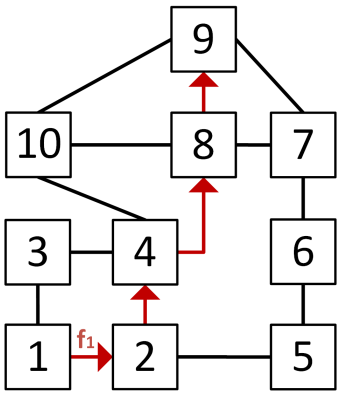

Table de routage nominale du nœud 4

id(f)	Route
1	2
2	3
3	4
5	0
6	3

Table de routage statique du nœud 4

Dest.	Route 1	Route 2	Route 3
1	3	4	-
2	3	-	-
3	4	-	-
5	3	2	-
6	3	2	1
7	2	1	3
8	2	-	-
9	2	1	3
10	1	-	-

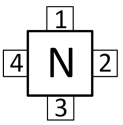

FIGURE 3.4 – Tables de routage nominale et statique

donnée. L'utilisation d'une étiquette, ici l'identifiant de flux, plutôt qu'une adresse plus longue, accélère la commutation. C'est un principe que l'on retrouve dans le protocole à commutation d'étiquettes MPLS dans l'Internet [Rosen et al., 2001]. La Figure 3.4 donne un exemple simple de la table de routage d'un nœud SDRN pour les flux temps réel. Lorsqu'un paquet du flux f_1 arrivera sur le nœud N_4, il sera commuté sur le port 2, selon la table de routage nominale. Notons qu'il serait tout à fait possible d'embarquer le chemin dans les entêtes des paquets au lieu de distribuer l'information. Dans l'exemple précédent, le chemin {2-1-2-1-0}, suite des ports à traverser, serait embarqué dans les entêtes des paquets du flux f_1. Chaque routeur dépilerait le numéro de port le plus à gauche avant de commuter le paquet vers celui-ci.

Enfin, puisque le contrôleur tient compte de l'état du réseau lors de la recherche de chemin, le routage en mode nominal est dynamique. En résumé, le routage des flux temps réel en mode nominal est un routage dynamique, à la source et orienté donnée.

Communications best-effort

Les communications *best-effort* ne passent pas par le contrôle d'admission et peuvent utiliser librement le canal dédié (Figure 3.2). Le chemin qu'elles suivent n'est donc pas calculé *a priori* ni optimisé par le contrôleur. Elles sont routées saut par saut jusqu'au nœud destination, dont l'adresse est portée par le paquet, de la même façon que dans un réseau IP. Des tables de routage statiques sont calculées à l'initialisation par le contrôleur et configurées sur les nœuds. Plusieurs routes peuvent éventuellement être disponibles vers une destination donnée, ce qui permet de contourner les pannes. Par exemple sur la Figure 3.4, si un paquet *best-effort* ayant pour destination N_7 arrive sur le nœud N_4, la route 1 de la table de routage statique est suivie : le paquet est commuté vers le port 2. Si ce lien est indisponible, alors la route 2 (port 1) sera suivie, et ainsi de suite.

Enfin, pour éviter les boucles, un commutateur ne peut retourner un paquet au nœud duquel il provient que s'il n'a pas de route alternative vers la destination. Dans l'exemple précédent, si un paquet provient du port 2, alors la route 1 sera ignorée. Ainsi, le paquet ne bouclera pas entre N_4 et N_8 et empruntera un autre chemin qui lui permette d'arriver à destination. Ce mécanisme est similaire au mécanisme de *split horizon* utilisé dans Routing Information Protocol (RIP) pour éviter les boucles aux annonces de routes [Malkin, 1998].

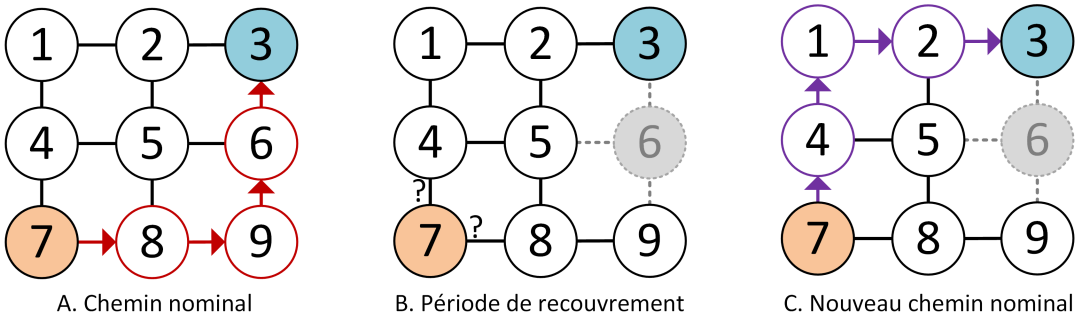

FIGURE 3.5 – Reconfiguration d'un flux suite à une panne d'un élément intermédiaire

3.4 Problématique de la tolérance aux fautes

3.4.1 Reconfiguration des flux

Dans le domaine des réseaux, la tolérance aux défaillances est généralement divisée en deux thèmes de recherche :

— Tolérance aux défaillances transitoires : consiste à gérer les éventuelles erreurs de transmission sporadiques. Un paquet peut être corrompu à cause d'une erreur de bit, il faut alors le retransmettre et assurer la conservation des garanties temps réel. Puisque l'erreur de transmission est sporadique, il n'y a pas de problématique de routage ou de contournement de la panne ;

— Tolérance aux défaillances permanentes : un nœud ou lien tombe en panne pour le reste de la durée de vie du réseau. Cette panne doit être gérée ou anticipée de façon efficace. Notons que la perte d'un nœud est équivalente à la perte des liens auxquels il est connecté, d'un point de vue du réseau.

Ce chapitre se concentre sur la **tolérance aux défaillances permanentes**, autrement appelée tolérance aux fautes. Le Chapitre 4 étudie la tolérance aux défaillances transitoires.

Dans le cas de la panne d'un nœud, le concepteur du système doit assurer que les applications critiques s'exécutant sur ce nœud pourront continuer à s'exécuter sur un autre, soit par un mécanisme de redondance passive ou active, soit par un mécanisme de migration à chaud. La contribution de ce chapitre se concentre sur le routage permettant de tolérer les pannes d'éléments intermédiaires entre la source et la destination.

Dans un réseau SDRN, la tolérance aux pannes des éléments intermédiaires est réalisée par la reconfiguration des flux affectés par la panne. Par exemple sur la Figure 3.5.A, le flux f_1 (rouge) est un flux temps réel critique. Le chemin sélectionné par le contrôleur durant le contrôle d'admission est le chemin : $path_{f_1} = \{7\text{-}8\text{-}9\text{-}6\text{-}3\}$. Dans cet exemple, le nœud 6 tombe en panne. Nous faisons l'hypothèse qu'il existe, à bas niveau, un mécanisme de détection de la panne qui va en informer les voisins puis le contrôleur SDRN. Ce dernier va alors reconfigurer le réseau, de sorte que f_1 suive un nouveau chemin sûr. S'il n'existe pas de tel chemin, à cause de la topologie ou de la charge du réseau, le contrôleur doit prendre des décisions supplémentaires pour libérer les ressources nécessaires à la réallocation de f_1. Il peut s'agir, par exemple, de reconfigurer les chemins empruntés par d'autres flux ou de modifier les paramètres protocolaires. Dans tous les cas, le contrôleur est capable, à terme, de garantir à nouveau le respect des contraintes temps réel de f_1 (Figure 3.5.C).

Nous étudions maintenant la période entre ces deux états nominaux, c'est-à-dire après que la panne ait eu lieu, mais avant que le flux ait été reconfiguré (Figure 3.5.B). Cette période transi-

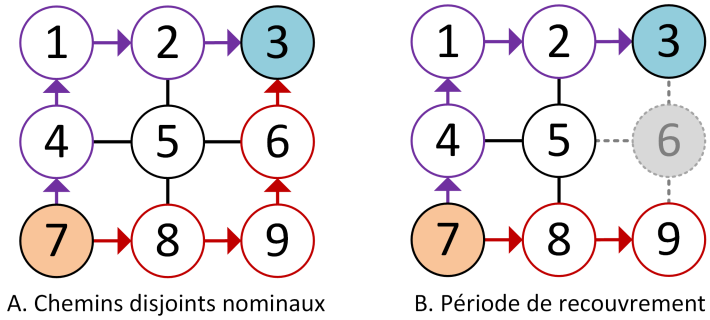

FIGURE 3.6 – Approche par chemins disjoints

toire est appelée **période de recouvrement de défaillance**. Certains flux les plus critiques ne peuvent tolérer la perte d'un seul paquet. Cela signifie que leurs contraintes temps réel doivent continuer à être garanties durant la période de recouvrement de défaillance. Notons de plus que, dans un réseau SDRN, il est très difficile de calculer la durée pire cas de la période transitoire. En effet, comme indiqué plus haut, la reconfiguration d'un flux peut impliquer la reconfiguration ou dégradation d'autres flux, ce processus est donc très complexe à borner. Nous souhaitons donc être capables d'assurer le respect des contraintes temps réel durant la période transitoire quelque soit sa durée.

A des fins de simplification, nous faisons l'hypothèse qu'une panne ne peut survenir durant la période de recouvrement de défaillance d'une autre. Cela signifie qu'il ne peut y avoir plusieurs pannes dans un intervalle de temps très réduit.

3.4.2 Approches classiques

Nous avons identifié deux approches naïves permettant d'aborder le problème de la tolérance aux pannes dans les réseaux temps réel. La première, que nous appellerons **approche par chemins disjoints**, consiste à calculer et réserver un chemin disjoint en parallèle du chemin nominal (Figure 3.6). Ce chemin disjoint doit bien sûr respecter les contraintes du flux. De cette façon, il est assuré que, quelque soit l'élément intermédiaire sur lequel survient la panne, il existe un chemin permettant aux paquets d'arriver à destination à temps, le temps qu'un nouveau chemin disjoint soit calculé. Ce principe est largement utilisé pour la tolérance aux pannes, par exemple dans les réseaux avioniques AFDX/ARINC-664p7, Parallel Redundancy Protocol dans l'automatisation de sous-stations (IEC 61850-9-2), ainsi que les réseaux TSN (Time-Sensitive Networks) [TSN Task Group, 2016e]. Cependant, cette approche nécessite la duplication des paquets à l'émission pour tolérer la phase de recouvrement de défaillance. En effet, si la panne survient au milieu de la transmission d'un paquet sur l'un des deux chemins, il n'est pas garanti qu'il puisse revenir en arrière jusqu'à la source pour emprunter le chemin disjoint, qui plus est en respectant sa contrainte d'échéance. Cette nécessité de duplication implique une utilisation massive des ressources réseau par rapport à ce qui est effectivement nécessaire en mode nominal. De plus, elle amène une problématique supplémentaire de gestion des doublons à la réception.

Une deuxième approche naïve, dite **par sous-chemins**, consiste à prévoir un sous-chemin de secours pour chaque panne intermédiaire possible, qui permette au paquet d'arriver à destination à temps (Figure 3.7). Cette approche ne nécessite pas de duplication des paquets car ces derniers partent directement du nœud précédant la panne pour emprunter le sous-chemin de secours. Éventuellement, seul un délai minime de changement de sous-chemin doit être pris en compte

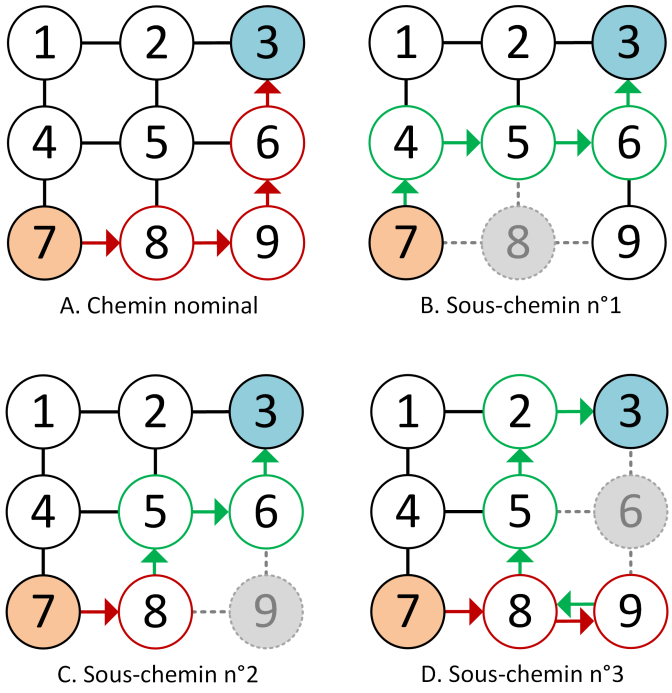

FIGURE 3.7 – Approche par sous-chemins

lors de la phase d'analyse des contraintes du contrôle d'admission. En revanche, le surcoût causé par le calcul et la configuration de tous les sous-chemins est important.

Étonnamment, la problématique de la période transitoire de reconfiguration dans un contexte temps réel est, à notre meilleure connaissance, très peu étudiée dans la littérature.

3.5 HYROS : Hybrid Routing in SDRN

3.5.1 Principe général

Notre contribution consiste à aborder la problématique de la période transitoire de recouvrement de défaillance de façon plus optimisée que les approches classiques que nous venons de décrire. Pour ce faire, nous proposons une combinaison de deux paradigmes de routage : routage à la source orienté donnée en mode nominal, et saut par saut orienté machine en période de recouvrement. En mode nominal, le contrôleur SDRN maîtrise et optimise l'utilisation du réseau en définissant les routes pour chaque flux. Cela permet de réduire les ressources réseau réservées au strict nécessaire. En revanche, lorsqu'un paquet rencontre une panne sur son chemin, il déclenche le mode de recouvrement pour lui-même. Un paquet en mode recouvrement est routé jusqu'à la destination en saut par saut, en fonction du nœud destination au lieu de l'identifiant de flux. Ce paradigme offre une grande flexibilité afin de s'adapter à chaque panne possible, sans nécessiter le calcul de sous-chemins de secours dédiés. En revanche, il requiert de répondre à deux questions :

— Comment garantir qu'un paquet en mode recouvrement pourra être routé jusqu'à la destination sans calculer le chemin exact qu'il va suivre ?

— Comment s'assurer que les ressources réseau seront suffisantes pour accepter les flux en mode recouvrement dans le pire cas ?

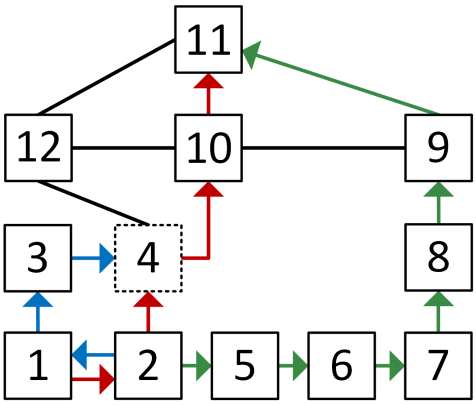

FIGURE 3.8 – Routes statiques alternatives

Nous définissons comme résilient un chemin dont les paquets sont assurés de rejoindre leur destination à temps en cas de panne d'un élément intermédiaire. Répondre aux deux questions ci-dessus implique de définir une méthode de garantie de la résilience d'un chemin.

3.5.2 Routage en mode recouvrement

En mode recouvrement, les paquets ne sont pas routés selon un chemin prédéfini par le contrôleur, mais en saut par saut jusqu'au nœud destination. Pour ce faire, les commutateurs utilisent les tables de routage statiques calculées à l'initialisation pour les flux non temps réel (cf. 3.3.2). La mise en place du routage en mode recouvrement est donc automatique à l'initialisation et ne nécessite pas de calcul à l'exécution (hors mise à jour des tables de routage en cas de changement de topologie du réseau).

Plusieurs routes peuvent éventuellement exister vers une destination donnée. Le cas échéant, la route de numéro $i+1$ est sélectionnée si et seulement si la route de numéro i commute le paquet vers un voisin (lien ou nœud) en panne. De plus, afin de s'assurer qu'une route alternative ne route pas le paquet vers la panne ayant causé la sélection de ladite route alternative, on retire du modèle du réseau les nœuds correspondant aux routes précédentes avant de la calculer. Par exemple sur la Figure 3.8, le nœud N_2 commute les paquets vers le nœud N_4 pour rejoindre N_{11} (chemin rouge). Si on ne retire pas N_4 du modèle lors du calcul de la route alternative, celle-ci passera par N_1, ce qui routera le paquet vers un cul-de-sac en N_3 (chemin bleu). Le paquet devra alors faire demi-tour, ce qui implique un délai à prendre en compte. En retirant N_4 du graphe *a priori*, la route alternative passera par N_5 et contournera parfaitement la panne (chemin vert). Notons qu'un commutateur ne peut savoir si un voisin est indisponible parce que le lien est défaillant ou parce que le nœud entier est défaillant, c'est pourquoi on prend le pire cas lors du calcul de la route alternative, à savoir la défaillance du nœud. Une exception à cette règle est lorsque le voisin est la destination. Puisque cela n'a pas de sens de router un paquet vers une destination en panne, seul le lien adjacent est retiré du modèle du réseau lors du calcul de la route alternative. On rappelle que la panne du nœud destination n'est pas traitée dans ce chapitre, et doit l'être par un mécanisme de redondance séparé.

L'Algorithme 8 synthétise le calcul des routes statiques à l'initialisation. Ces routes sont ensuite distribuées aux nœuds par l'interface de configuration avec le matériel. Le contrôleur conserve l'information globale, qui sera utilisée pour vérifier la résilience des chemins.

Chaque paquet de flux temps réel emporte à la fois son identifiant, id_f, et l'adresse du nœud

Algorithme 8 : Calcul des tables de routage statiques à l'initialisation

```
pour chaque nœud ∈ réseau faire
    réseau_temp ← réseau
    pour chaque autre_nœud ∈ (réseau\nœud) faire
        port ← plus_court_chemin(nœud, autre_nœud)
        si port est non null alors
            nœud.ajouter_route(autre_nœud, port)
            retirer nœud_voisin(port) de réseau_temp
            continuer
        fin
        sinon
            sortir de la boucle
        fin
    fin
fin
```

destination, $dest_f$. Si le commutateur ne peut suivre la route nominale selon id_f à cause de la défaillance du lien ou nœud voisin, il déclenche le mode recouvrement pour le paquet. Un drapeau est positionné à 1 dans l'entête du paquet. Cette information permet aux commutateurs de savoir que le paquet est en mode recouvrement et qu'il ne faut pas le jeter. Elle est également utilisée pour l'ordonnancement au niveau du port de sortie (cf. Section 3.5.4). Lorsque le paquet rejoint son chemin nominal (dans le pire cas, au niveau de la destination), il peut quitter le mode recouvrement.

3.5.3 Caractérisation de la résilience des chemins

Le mécanisme de routage hybride que nous proposons avec HYROS n'est viable pour un flux que s'il est possible d'assurer *a priori* qu'il existera toujours une route permettant aux paquets du flux de rejoindre leur destination à temps en mode recouvrement. Ce processus s'appelle la caractérisation de la résilience des chemins et s'effectue lors du contrôle d'admission. De plus, cela doit être réalisé sans calculer les routes de recouvrement effectives, afin de valoriser les tables de routage déjà calculées et réduire au maximum le surcoût de cette opération.

Pour qu'un chemin soit résilient, il faut qu'il existe au moins une route alternative vers la destination depuis chaque nœud qui le compose. Puisque le chemin nominal consomme une route, il est donc résilient seulement s'il existe au moins deux routes vers la destination depuis chaque nœud qui le compose (l'une d'entre elles pouvant être la route nominale et donc indisponible en mode recouvrement).

De plus, la route alternative doit permettre aux paquets de rejoindre leur destination à temps. Il existe donc une contrainte sur la longueur des routes alternatives : le nombre de sauts requis pour rejoindre la destination doit être inférieur à la contrainte de longueur de chemin, moins la longueur du chemin parcouru. Pour caractériser le respect de cette contrainte, on ajoute aux tables de routage statique le coût des routes, c'est-à-dire le nombre de sauts requis pour rejoindre la destination. On parle aussi de vecteurs de distance. Ces coûts peuvent être facilement calculés lors de la phase d'initialisation, puisque les routes statiques sont déduites de chemins calculés par le contrôleur, depuis chaque nœud vers chaque autre nœud du réseau. La Figure 3.9 illustre une table de routage statique à laquelle on a ajouté les vecteurs de distance ou coûts entre

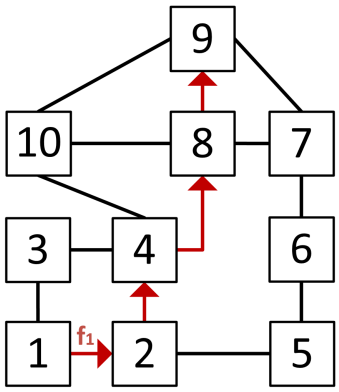

Table de routage statique du nœud 4

Dest.	Route 1	Route 2	Route 3
1	3 (2)	4 (2)	-
2	3 (1)	-	-
3	4 (1)	-	-
5	3 (2)	2 (4)	-
6	3 (3)	2 (3)	1 (4)
7	2 (2)	1 (3)	3 (4)
8	2 (1)	-	-
9	**2 (2)**	**1 (2)**	**3 (5)**
10	1 (1)	-	-

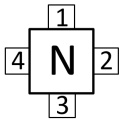

FIGURE 3.9 – Routes statiques avec vecteurs de distance

parenthèses. Une route indisponible a implicitement un coût infini.

Ainsi, un chemin est résilient seulement si, pour chaque nœud qui le compose (hors destination), il existe au moins deux routes vers le nœud destination dont le coût est inférieur ou égal à la contrainte de taille de chemin, moins la position du nœud étudié dans le chemin.

Par exemple sur la Figure 3.9, on veut caractériser la résilience du chemin $path_{f_1} = \{1\text{-}2\text{-}4\text{-}8\text{-}9\}$. La contrainte de taille de chemin est 6. $path_{f_1}$ est résilient seulement si :

— N_1 possède au moins deux routes vers N_9, dont le coût est inférieur ou égal à 5 ;
— N_2 possède au moins deux routes vers N_9, dont le coût est inférieur ou égal à 4 ;
— N_4 possède au moins deux routes vers N_9, dont le coût est inférieur ou égal à 3 ;
— N_8 possède au moins deux routes vers N_9, dont le coût est inférieur ou égal à 2.

Si N_4 n'était pas relié à N_{10}, alors $path_{f_1}$ ne serait pas résilient, car la route alternative de N_4 vers N_9 serait trop longue (route 3, 5 sauts).

Cette étape de vérification, décrite par l'Algorithme 9, consiste à regarder $(\|path_f\| - 1)$ (dans cet exemple, quatre) tables de routage statiques, ce qui est très rapide et consiste en un surcoût négligeable, permettant son utilisation à l'exécution. Elle a également l'avantage de valoriser les tables de routage calculées à l'initialisation, ainsi que l'étape de traduction des contraintes temps réel en taille maximale de chemin effectuée lors du contrôle d'admission (Figure 3.3). L'analyse temporelle d'une requête n'est effectuée qu'une seule fois dans toute sa complexité, puis l'étape de recherche de chemin se base (entre autres) sur cette contrainte très simple de taille maximale de chemin. On est alors capable de caractériser la résilience d'un chemin sans connaître les routes alternatives qui seront suivies en mode recouvrement.

3.5.4 Canal de recouvrement

La disponibilité de routes alternatives est une condition nécessaire mais pas suffisante à la garantie du respect des contraintes temps réel en mode recouvrement. Il faut également s'assurer que suffisamment de sous-canaux seront disponibles pour le flux en mode recouvrement sur les routes alternatives, toujours sans calculer les routes effectives. Ceci est nécessaire pour que le flux respecte sa contrainte de bande passante. C'est également nécessaire pour assurer que les paquets du flux ne seront pas retardés et que les coûts des routes utilisés lors de l'étape précédente sont fiables en termes de respect des contraintes d'échéances. En effet, la traduction des contraintes temps réel en taille maximale de chemin suppose que le flux est stable sur tous les liens par lesquels il passe.

Pour conserver la flexibilité et efficience apportées par le routage hybride, nous proposons

Algorithme 9 : Algorithme de caractérisation de la résilience d'un chemin

```
1  Fonction vérifier_resilience(chemin, taille_max_f)
2      destination ← chemin[chemin.taille]
3      pour i de 1 à (chemin.taille - 1) faire
4          nœud_courant ← chemin[i]
5          si nœud_courant.route1(destination).coût ≤ (taille_max_f − i) ET
             nœud_courant.route2(destination).cost ≤ (taille_max_f − i) alors
6              continuer
7          fin
8          sinon
9              retourner faux
10         fin
11     fin
12     retourner vrai
```

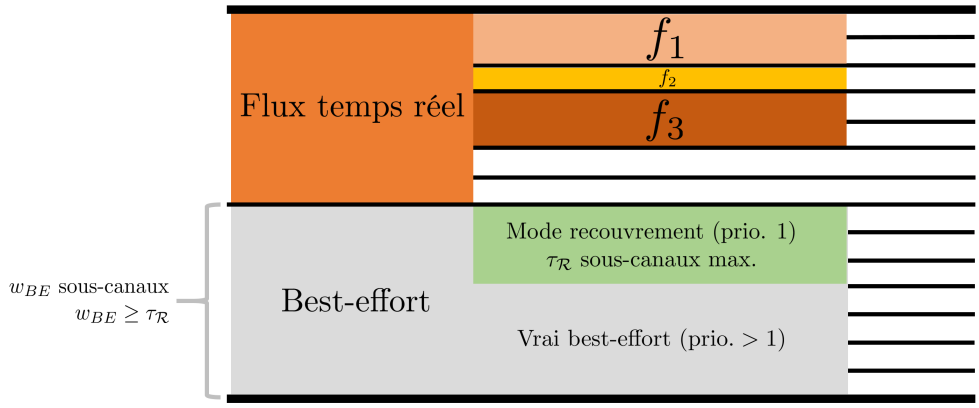

FIGURE 3.10 – Canal de recouvrement

que les communications en mode recouvrement utilisent le canal *best-effort*, avec une priorité supérieure à celle des flux réellement *best-effort* (Figure 3.10). De cette façon, le canal *best-effort* se comporte comme un canal de secours partagé, sans être dédié aux communications en mode recouvrement. Puisque les paquets en mode recouvrement ont un niveau de priorité supérieur, ils ne seront pas retardés par la présence de paquets *best-effort*. Ils pourront ainsi utiliser jusqu'à la totalité du canal *best-effort* en cas de besoin. Il existe cependant une exception à cette affirmation : s'il n'existe pas de mécanisme de préemption des paquets à bas niveau, un paquet en mode recouvrement peut subir un délai correspondant à l'envoi d'un paquet de taille maximale dans le pire cas. C'est le cas lorsqu'un paquet en mode recouvrement est placé dans le tampon de sortie alors qu'un paquet *best-effort* vient de démarrer sa transmission. Ce délai doit être pris en compte lors de l'étape d'analyse temporelle (cf. Section 3.5.5).

Le nombre de sous-canaux virtuels requis pour les flux en mode recouvrement peut cependant dépasser la largeur total du canal *best-effort*. Il faut donc s'assurer que, dans le pire cas, le poids total des flux en mode recouvrement ne dépassera pas la largeur du canal *best-effort*. L'un des avantages d'HYROS est le fait qu'il ne nécessite pas le calcul des routes de recouvrement effectives. Cependant, cela implique que, dans le pire cas, nous devons considérer que tous les flux en mode recouvrement passent par le même lien. C'est essentiellement pour cette raison que

le routage utilisé en mode recouvrement ne serait pas efficace en mode nominal, d'où l'intérêt d'une combinaison de deux paradigmes différents.

Lorsqu'un nœud tombe en panne, tous les flux **transitant** par ce nœud basculeront en mode recouvrement. Les flux dont c'est la destination ne basculeront pas en mode recouvrement car, comme postulé plus haut, ce dernier ne gère pas les pannes des nœuds destination. Le pire cas correspond donc à la panne d'un nœud dont le total des poids des flux transitant par ce nœud est maximal. Cette valeur est notée $\tau_{\mathcal{R}}$. Dans tout notre raisonnement, la lettre τ sera associée à la tolérance aux pannes. Elle n'a aucun rapport avec le symbole utilisé pour représenter les tâches en théorie de l'ordonnancement.

On définit :

— $\mathcal{F}_{\mathcal{R},\tau}$ l'ensemble des flux ayant accès au mode recouvrement dans le réseau. Il s'agit des flux qui ne peuvent accepter de rupture de communication temporaire durant la phase de recouvrement ;

— $w_{f,LE}$ le poids du flux f sur les liens externes ;

— N un nœud du réseau ;

— $\mathcal{R}$ le réseau.

Le poids total pire cas des flux en mode recouvrement, $\tau_{\mathcal{R}}$, est donc calculé comme suit :

$$\tau_{\mathcal{R}} = max_{N \in \mathcal{R}} \left(\sum_{f \in \mathcal{F}_{\tau}} w_{f,LE} \middle| N \in (path_f \setminus dest_f) \right) \tag{3.1}$$

Cette valeur doit être inférieure à la largeur du canal *best-effort* sur les liens externes, $w_{BE,LE}$ (Figure 3.10). Si cette condition n'est pas respectée, alors il faut réduire l'ensemble $\mathcal{F}_{\mathcal{R},\tau}$ puis réitérer. Le ou les flux retiré(s) de $\mathcal{F}_{\mathcal{R},\tau}$ doivent être dupliqué(s) le long de chemins disjoints pour assurer leur résilience. Notons que ce cas n'apparaît que lorsqu'il y a trop de flux autorisés à utiliser le mode recouvrement et/ou que le canal *best-effort* est sous-dimensionné. Si le concepteur du système souhaite éviter la duplication de flux sur des chemins disjoints, alors il doit exister suffisamment de ressources pour le mode recouvrement. Cela signifie que la largeur du canal *best-effort* doit être réglée en cohérence avec les besoins potentiels du mode recouvrement. Plusieurs heuristiques sont possibles pour optimiser la régulation de $\mathcal{F}_{\mathcal{R},\tau}$. Une première solution est de systématiquement retirer le flux le plus volumineux de $\mathcal{F}_{\mathcal{R},\tau}$, parmi ceux transitant par le nœud le plus chargé. Une autre idée est de ne pas attendre que $\tau_{\mathcal{R}}$ soit trop grand, en commençant à retirer et dupliquer des flux dès que $\tau_{\mathcal{R}}$ atteint 66% ou 75% de $w_{BE,LE}$, de sorte à lisser ce mécanisme.

L'Algorithme 10 est un algorithme gourmand de calcul de $\tau_{\mathcal{R}}$, suivant l'équation 3.1. Afin d'optimiser ce calcul et permettre son passage à l'échelle, nous proposons de ne pas recalculer $\tau_{\mathcal{R}}$ de zéro à chaque fois. Pour ce faire, on conserve en mémoire des valeurs de poids pour chaque nœud du réseau. Ces valeurs sont mises à jour de façon incrémentale à chaque allocation ou libération de flux (Algorithmes 11 et 12). Le calcul de $\tau_{\mathcal{R}}$ consiste alors à sélectionner la valeur maximale parmi l'ensemble des nœuds (Algorithme 13).

3.5.5 Délais additionnels

Le mode recouvrement peut faire subir aux flux des délais supplémentaires qui doivent être pris en compte lors de l'analyse temporelle. On rappelle que, dans notre raisonnement, la lettre τ est appliquée à tout élément impliqué dans la tolérance aux pannes. Un rappel de la table des symboles est donnée en Tableau 3.2 pour faciliter la lecture de la suite.

Algorithme 10 : Calcul de $\tau_{\mathcal{R}}$ (algorithme gourmand)

```
1  Fonction calculer_τ_R(réseau, F_τ)
2      τ_R ← 0
3      pour chaque nœud ∈ réseau faire
4          τ_R_temp ← 0
5          pour chaque flux ∈ F_τ faire
6              si flux.chemin.contient(nœud) ET nœud ≠ flux.destination alors
7                  τ_R_temp ← τ_R_temp + flux.poids
8              fin
9          fin
10         τ_R ← max(τ_R, τ_R_temp)
11     fin
12     retourner τ_R
```

Algorithme 11 : Augmentation de $\tau_{\mathcal{R}}$ (allocation de flux)

```
1  Fonction augmentation_τ_R(réseau, flux)
2      pour chaque nœud ∈ flux.chemin \ flux.destination faire
3          nœud.τ_N ← nœud.τ_N+flux.poids
4      fin
```

Algorithme 12 : Diminution de $\tau_{\mathcal{R}}$ (désallocation de flux)

```
1  Fonction diminution_τ_R(réseau, F_τ)
2      pour chaque nœud ∈ flux.chemin \ flux.destination faire
3          nœud.τ_N ← nœud.τ_N−flux.poids
4      fin
```

Algorithme 13 : Calcul de $\tau_{\mathcal{R}}$ (algorithme incrémental)

```
1  Fonction calculer_τ_R(réseau, F_τ)
2      τ_R ← 0
3      pour chaque nœud ∈ réseau faire
4          si nœud.τ_N > τ_R alors
5              τ_R ← nœud.τ_N
6          fin
7      fin
8      retourner τ_R
```

TABLE 3.2 – Table des symboles

Symbole	Définition	Moyen mnémotechnique
T_f	Période minimale d'envoi de donnée du flux f	Symbole de période
S_f	Taille maximale de donnée du flux f	Size
dl_req_f	Échéance relative du flux f	Deadline requirement
C_L	Débit utile maximal sur le lien L	Capacité
Ch_L	Nombre de sous-canaux virtuels sur le lien L	Channels
R_L	Nombre de sous-canaux virtuels vacants sur le lien L	Remaining
$\mu_{f,L}$	Taille des paquets de tête d'une donnée du flux f sur le lien L	MTU
$\epsilon_{f,L}$	Taille du paquet de queue d'une donnée du flux f sur le lien L	Symbole de reliquat
ψ_L	Charge utile maximale sur le lien L	Payload size
$\psi'_{f,L}$	Limite à la charge utile pour le flux f sur le lien L	Payload size
η_L	Taille d'entête maximale sur le lien L	Header
$\phi_{f,L}$	Nombre maximal de fragments pour une donnée du flux f sur le lien L	Fragments
$\theta_{f,L}$	Taille totale d'une donnée du flux f sur le lien L après fragmentation	Total size
$w_{f,L}$	Poids minimum requis du flux f sur le lien L	Weight
$d_{sw,\mathcal{R}}$	Latence de commutation (constante)	
$d_{f,L}$	Délai de saut du flux f sur le lien L	
$\beta_{f,L}$	Délai de rafale du flux f sur le lien L	Burst delay

Premièrement, si aucun mécanisme de préemption n'est disponible au niveau du module d'ordonnancement, il peut survenir des inversions de priorité dans le canal *best-effort* : un paquet en mode recouvrement est retardé par un paquet *best-effort* moins prioritaire dont la transmission avait déjà débuté. Le délai subi est dans le pire cas égal au temps de transmission d'un paquet de taille maximale. Il doit être pris en compte dans le délai de saut sur les liens externes (cf. Section 2.5.1). On ne considère ce problème que sur les liens externes, car les liens internes de la source et de la destination fond obligatoirement partie du chemin nominal. Un flux ne peut donc pas se retrouver en mode recouvrement sur un lien interne. Le délai de saut étendu, noté $d^{\tau}_{f,LE}$, est donc calculé ainsi :

$$d^{\tau}_{f,LE} = \frac{\mu_{f,L} + (Ch_L - w_{f,L}) \cdot Q_L}{C_L} + \frac{\boldsymbol{\psi_{LE} + \eta_{LE}}}{\boldsymbol{C_{LE}}} \tag{3.2}$$

Deuxièmement, le temps de détection de la panne d'un élément voisin peut être non négligeable. Le cas échéant, il peut être subi une fois par un paquet en mode recouvrement, il est donc pris en compte dans le délai de rafale. Le délai de rafale étendu, noté $\beta^{\tau}_{f,\mathcal{R}}$, est calculé ainsi :

$$\beta^{\tau}_{f,\mathcal{R}} = \max_{\{LI,LE\}} \left(\frac{\theta_{f,L} - \mu_{f,L}}{C_L} + \left\lceil \frac{\theta_{f,L}}{\lfloor w_{f,L} \cdot Q_L / \mu_{f,L} \rfloor \cdot \mu_{f,L}} - 1 \right\rceil \cdot \frac{(Ch_L - w_{f,L}) \cdot Q_L}{C_L} \right) + \boldsymbol{t_{détection}} \tag{3.3}$$

L'analyse de résilience des chemins doit alors se baser sur une latence pire cas étendue, notée $wc\_délai^{\tau}(f, path_f)$. Elle utilise les délais de saut et de rafale étendus explicités ci-dessus.

On obtient alors l'équation 3.4 :

$$wc\_délai^{\tau}(f, path_f) = 2 \cdot d_{f,LI} + (\|path_f\| - 1) \cdot d^{\tau}_{f,LE} + \|path_f\| \cdot d_{sw,\mathcal{R}} + \beta^{\tau}_{f,\mathcal{R}} \tag{3.4}$$

La taille maximale de chemin qui doit être utilisée lors de la phase de caractérisation de résilience est donc :

$$taille\_max^{\tau}_f = \frac{dl\_req_f + d^{\tau}_{f,LE} - 2 \cdot d_{f,LI} - \beta^{\tau}_{f,\mathcal{R}}}{d_{sw,\mathcal{R}} + d^{\tau}_{f,LE}} \tag{3.5}$$

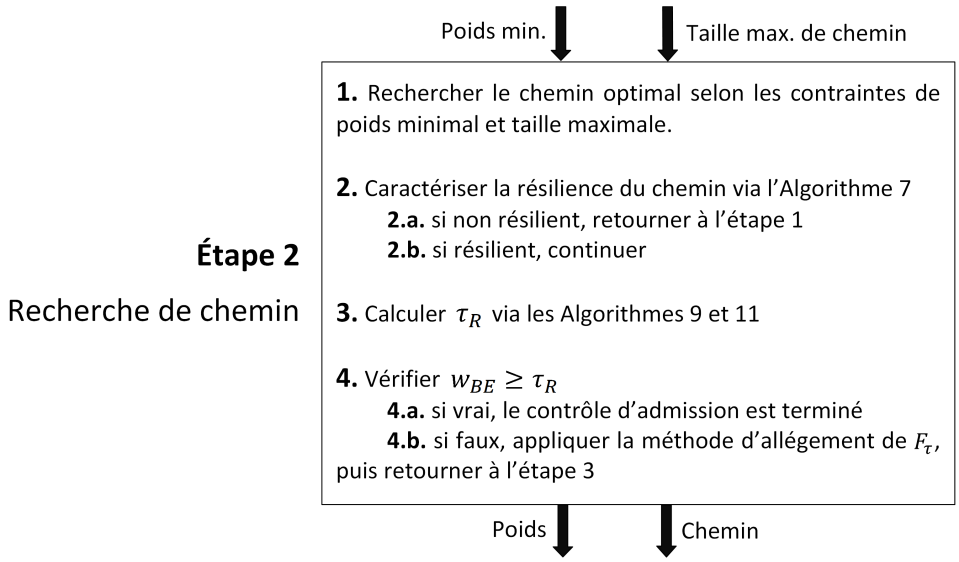

FIGURE 3.11 – Extension de l'étape de recherche de chemin lors du contrôle d'admission

3.5.6 Accès au mode recouvrement

Les flux souhaitant avoir accès au mode recouvrement, c'est-à-dire qui ne peuvent accepter de rupture de communication temporaire durant la phase de reconfiguration, doivent le déclarer dans leur requête de flux. Un drapeau, τ_f, porte cette information. Les requêtes de flux SDRN deviennent donc : $requête_f = \{type_f, T_f, S_f, dl\_req_f, src_f, dest_f, \boldsymbol{\tau_f}\}$.

3.5.7 Synthèse du mode recouvrement

HYROS est un algorithme de routage hybride permettant de gérer la période transitoire de recouvrement de défaillance de façon efficiente dans un réseau SDRN. Il étend l'architecture SDRN à quatre niveaux :

— A l'initialisation, le contrôleur SDRN calcule des tables de routage statiques avec vecteurs de distance ;
— Les flux demandant la tolérance aux pannes doivent l'indiquer en requête de flux via un drapeau τ_f ;
— Au contrôle d'admission, la résilience des chemins doit être vérifiée pour les flux demandant la tolérance aux pannes ;
— A l'exécution, les commutateurs doivent déclencher et suivre le mode recouvrement si nécessaire.

Lorsqu'elles requièrent l'allocation d'un flux temps réel, les applications doivent indiquer si le flux demande l'accès au mode recouvrement. C'est le cas lorsque le flux ne peut accepter de rupture temporaire de communication le temps de la période de recouvrement. Si oui, l'algorithme de recherche de chemin (Figure 3.3, étape 2) doit être étendu afin de :

— Vérifier que le chemin est résilient ;
— Vérifier que l'inéquation $w_{BE,LE} \geq \tau_{\mathcal{R}}$ est toujours respectée si le flux est alloué sur le chemin étudié.

Cette extension est résumée en Figure 3.11.

Dès que le nœud détecte la panne d'un élément voisin, il notifie le contrôleur SDRN et déclenche le mode recouvrement pour les paquets qui le nécessitent. Lorsque le contrôleur SDRN

TABLE 3.3 – Temps moyen du contrôle d'admission dans un réseau tore de 25 nœuds

Approche	**Base**	**+ Résilience**
HYROS	1,04 ms	1,56 ms
Chemins disjoints	1,04 ms	2,40 ms
Sous-chemins	1,04 ms	31,10 ms

reçoit la notification de la panne, il met à jour son modèle du réseau, les tables de routage statiques, et reconfigure les flux affectés par la panne (éventuellement en dialoguant avec le module applicatif). Une fois ce processus terminé, une panne peut à nouveau être tolérée de façon sûre. Cela signifie qu'HYROS est capable de tolérer un nombre infini de pannes, à condition qu'elles n'apparaissent pas durant la même période de recouvrement (dans la version actuelle). De plus, HYROS n'impose aucune contrainte sur la durée de la période de recouvrement. Celle-ci peut durer le temps nécessaire pour reconfigurer les flux.

3.6 Évaluation

3.6.1 Protocole d'évaluation

Nous réutilisons le protocole d'évaluation décrit en Chapitre 2 : implémentation en Java 1.8 du contrôleur, sur ordinateurs Raspberry Pi 3, dans le cas d'un réseau tore de 25 nœuds (5X5). La recherche de chemin utilise la méthode hybride 25/75. Nous étendons l'implémentation du contrôle d'admission avec :

— HYROS, c'est-à-dire l'algorithme de caractérisation de résilience synthétisé en Section 4 ;
— Une approche par chemins disjoints. Dans cette approche, un chemin est résilient si et seulement si il existe un autre chemin disjoint de la source à la destination, qui respecte les contraintes temps réel du flux. En pratique, on retire du graphe les arcs et nœuds intermédiaires qui composent le chemin étudié, puis on regarde s'il existe un chemin entre la source et la destination. Si oui, alors le chemin étudié est résilient ;
— Une approche par sous-chemins. Dans cette approche, un chemin est résilient si et seulement si, pour chaque élément intermédiaire, il existe un sous-chemin permettant d'arriver à destination à temps sans transiter par cet élément intermédiaire.

Dans notre expérimentation, tous les flux requièrent la garantie de la tolérance aux pannes.

3.6.2 Évaluation quantitative

On évalue tout d'abord le temps moyen nécessaire au contrôle d'admission. Les résultats sont donnés dans le Tableau 3.3.

Avec HYROS, le surcoût dû à la vérification de la résilience des chemins (Algorithme 9) est négligeable, car il consiste à regarder ($\|path_f - 1\|$) fois dans les tables de routage statiques. Le surcoût observé en Tableau 3.3 est dû au fait qu'un chemin a une chance plus élevée d'être refusé à cause de la contrainte de résilience additionnelle. Le contrôle d'admission parcourt donc plus de chemins en moyenne. De plus, en ajoutant la contrainte de résilience, il n'est plus possible de calculer rapidement si au moins une solution existe, ce qui augmente considérablement le temps de recherche de chemin lorsqu'il n'existe aucune solution. Toutefois, cette évaluation montre l'applicabilité d'HYROS à l'exécution.

Dans le cas de l'approche par chemins disjoints, la caractérisation de la résilience d'un chemin

TABLE 3.4 – Évaluation ordinale des approches de tolérance aux pannes

Approche	Surcoût Contrôle d'admission	Surcoût Ressources réseau	Optimalité des chemins
HYROS	+	++	++
Chemins disjoints	+	−−	+
Sous-chemins	−−	++	++

est également rapide, car elle correspond à une simple recherche de plus court chemin. Cependant, la contrainte de chemin disjoint est une contrainte de résilience plus forte que celle imposée par HYROS. Un chemin a donc plus de chances d'être refusé avec l'approche par chemins disjoints, d'où l'observation d'un surcoût légèrement plus élevé. Malgré ce surcoût, cette approche reste tout à fait utilisable à l'exécution. En revanche, elle requiert la réservation du double de ressources réseaux, et le chemin disjoint peut être un chemin peu optimal par rapport à la stratégie d'optimisation définie. Avec HYROS, les ressources utilisées en mode recouvrement ne sont utilisées qu'en cas de nécessité, et disponibles pour d'autres communications sinon.

Enfin, l'approche par sous-chemins impose un surcoût massif car la caractérisation de la résilience implique de nombreuses recherches de chemins. Cela met en valeur l'intérêt de notre approche, qui est capable de caractériser la résilience d'un chemin sans calculer les routes alternatives effectives. Ceci est possible par le changement de paradigme de routage, qui permet au mode recouvrement de se baser sur la connaissance commune que sont les tables de routages statiques.

3.6.3 Évaluation ordinale

Nous effectuons une évaluation ordinale des trois approches, selon trois critères (Tableau 3.4). Le vert foncé indique que la réponse au critère est parfaite, le vert clair qu'elle est bonne et le rouge qu'elle est mauvaise. Justifions les résultats de cette évaluation :

- Surcoût du temps de contrôle d'admission : cette évaluation ordinale découle des résultats du Tableau 3.3 ;
- Surcoût des ressources réseau utilisées : HYROS et l'approche par sous-chemins n'utilisent les ressources réseau que lorsqu'elles sont strictement nécessaires. L'approche par chemins disjoints requiert une duplication active des paquets ;
- HYROS et l'approche par sous-chemins permettent à plus de chemins d'être acceptés. En effet, la contrainte d'existence d'un chemin complet disjoint est plus forte. Cela signifie qu'un chemin refusé avec l'approche par chemins disjoints peut être accepté avec les deux autres, mais pas l'inverse. Plus précisément, la contrainte de résilience d'HYROS et de l'approche par sous-chemins est la plus faible possible. Elle ne demande que l'existence d'un chemin qui ne passe pas par la panne. En conclusion, l'approche par chemins disjoints retournera toujours des chemins autant ou moins optimisés. Elle reste cependant globalement satisfaisante car la contrainte d'existence de chemins disjoints n'est pas excessif dans un réseau à topologie résiliente, tel qu'un réseau maillé.

Nous pouvons globalement conclure de cette évaluation que le changement de paradigme de routage en mode recouvrement permet à HYROS de profiter des bonnes caractéristiques d'une approche par sous-chemins en termes d'utilisation du réseau, mais en réduisant drastiquement

la complexité d'analyse de résilience des chemins.

3.7 Routage hybride et tolérance aux pannes dans la littérature

La notion de routage hybride, à savoir la combinaison de paradigmes de routage, permet d'aborder un large panel de problématiques. De nombreuses recherches ont montré son intérêt pour apporter de la flexibilité à des réseaux optimisés, dans un objectif de performance, de passage à l'échelle ou de tolérance aux pannes.

Dans les réseaux ad hoc mobiles (MANETs) par exemple, combiner un routage proactif dans les zones et réactif entre les zones est une solution aux problématiques de passage à l'échelle [Raheja and Maakarn, 2014].

HYDRO [Dawson-Haggerty et al., 2010] est un protocole de routage hybride pour l'Internet des objets (IoT). Il combine un contrôle central des routes avec une agilité locale, toujours dans un objectif de passage à l'échelle. L'optimisation centralisée réduit le surcoût de routage vers les routeurs de bordure.

Les réseaux sur puce (Networks-on-Chip, NoCs) [Bjerregaard and Mahadevan, 2006], présentés dans le Chapitre 1, font face à des problématiques similaires aux nôtres en termes de routage et d'analyse de flux temps réel sur réseau maillé. D'un côté, les recherches abordent les problématiques d'analyse temporelle, dans le but de garantir des latences pire cas pour les flux temps réel, considérant le protocole de routage nominal et les possibilités de congestion. [Hesham et al., 2017] décrit certaines des techniques et défis pour la conception de réseaux sur puce temps réel. D'un autre côté, la tolérance aux pannes et plus particulièrement le routage tolérant aux pannes forment également un domaine de recherche important. L'objectif principal est la capacité à éviter le blocage de paquets dans le cas d'une panne de nœud, les interblocages et les boucles. A cette fin, le routage hybride a été montré dans de nombreuses recherches comme une solution prometteuse.

Duato [Duato, 1993, Duato, 1995, Duato, 1996] a théorisé le fait d'assurer que les règles de routage disposent toujours d'un canal virtuel vacant pour échapper aux interblocages. Dans HYROS, nous assurons que les communications en mode recouvrement seront toujours capables d'utiliser le canal *best-effort* en cas de besoin. Cependant, Duato n'aborde pas le problème de la validation des chemins de secours d'un point de vue temps réel.

Glass et Ni [Glass and Ni, 1996] est un autre algorithme de routage tolérant aux pannes historique. Il ne requiert pas l'utilisation de canal virtuel. Son inconvénient principal est le faible nombre de pannes qui peuvent être tolérées. [Fukushima et al., 2009] propose une solution à ce dernier problème pour des réseaux sur puce en deux dimensions.

[Boppana and Chalasani, 1995] propose un algorithme qui route les paquets suivant des chemins minimaux en mode nominal (pas de défaillance) et relâche cette contrainte pour contourner les pannes. Des anneaux de nœuds et liens non défaillants sont utilisés pour réaliser ces détours.

DyAD (*Dynamically Adaptive and Deterministic*) [Hu and Marculescu, 2004] combine un routage statique à faible charge réseau avec un routage adaptatif à haute charge. Le passage d'un mode à l'autre est déclenché par un drapeau de congestion. Dans HYROS, le changement de mode est également associé à un drapeau, mais flux par flux et non globalement. Un changement de mode global nécessite un contrôleur de modes qui coordonne les algorithmes utilisés [Wu et al., 2016], ce qui est un problème complexe. Dans HYROS, le changement de mode est décentralisé et autonome, il ne passe pas par le contrôleur SDRN.

[Kim and Kim, 2007] propose SRN, *Source Routing for Noc*. Ce protocole est composé de deux mécanismes de découverte de route et de maintenance de route. Ces deux mécanismes

collaborent pour permettre aux nœuds de construire et maintenir des routes vers des destinations arbitraires. La reconfiguration de route est effectuée sur demande. Cette approche est proche de SDRN, qui reconfigure les routes nominale en cas de panne. Cependant, dans SRN, la période de recouvrement est gérée par un mécanisme de redondance active, de façon similaire à l'approche par chemins disjoints que nous avons abordée dans ce chapitre. Comme expliqué en Section 3.6.3, ceci résulte en un surcoût massif en termes de ressources réseau utilisées.

XGFT (eXtended Generalized Fat Tree) [Kariniemi and Nurmi, 2005] est un réseau sur puce tolérant aux pannes, qui combine un routage saut par saut en mode nominal avec un routage déterministe pour éviter les pannes. Lorsqu'un paquet rencontre une défaillance, un mécanisme de diagnostic et réparation (*Fault-Diagnosis-And-Repair*, FDAR) reconfigure les routeurs, de sorte que le paquet soit routé un saut en arrière puis emprunte un chemin déterministe qui contourne la panne. Avec cette technique, le recouvrement de la défaillance ne peut démarrer avant la fin de l'exécution de l'algorithme FDAR. Cela signifie que, pour respecter des contraintes temps réel, il faudrait être capable d'analyser et garantir le temps pire cas d'exécution de l'algorithme FDAR, ce qui est très complexe et requiert une connaissance de l'état pire cas de l'état du réseau (trafic notamment). L'algorithme FDAR pourrait même conclure, à l'exécution, qu'il n'existe pas de solution permettant de contourner la panne. Dans HYROS, la résilience des flux temps réel est vérifiée avant le début des communications tout en étant agnostique de l'évolution de l'état du réseau. De plus, la durée de la période de recouvrement n'a pas besoin d'être garantie, car l'arrivée à temps des paquets est assurée par le mode recouvrement. La seule contrainte est que la période de recouvrement soit suffisamment courte pour qu'une deuxième panne ne puisse survenir avant la fin de la reconfiguration.

[Göhringer et al., 2011] montre l'intérêt de combiner les avantages de haute performance et faible latence d'un routage par circuit avec la flexibilité et passage à l'échelle d'un routage saut par saut, dans des objectifs d'optimisation mais également de tolérance aux pannes (évitement du blocage de paquets).

[Bishnoi, 2016] propose *Hybrid Fault-Tolerant Routing Algorithm* (HFTRA). Des canaux virtuels sont initialement configurés pour suivre des algorithmes de routage différents les uns des autres. Le changement de mode de routage est ainsi effectuée par un changement de canal virtuel. Cela permet une adaptation aux pannes très rapide. Cependant, ce système de garantit pas le respect des contraintes temps réel lorsqu'un algorithme de routage secondaire est utilisé.

Enfin, *Arbitration Look Ahead Scheme* (ALOAS) [Kim et al., 2005] utilise le même principe que SDRN en mode nominal. L'information de routage d'un flux est distribuée aux routeurs le long du chemin nominal avant le début de la communication. Cette méthode est revendiquée comme plus rapide qu'un routage à la source utilisant des chemins embarqués dans les entêtes, dû à la suppression du surcoût avant chaque envoi de paquet.

De manière générale, lorsqu'on étudie la littérature, on observe qu'il existe de nombreuses solutions de routage, hybrides ou non, pour gérer la tolérance aux pannes dans les réseaux sur puce maillés. Cependant, elles ne sont pas combinées avec les méthodes d'analyse temporelle qui font l'objet d'autres recherches. Ceci fait qu'il n'existe pas de solution de routage pour la tolérance aux pannes qui garantisse le respect des contraintes temps réel en période de recouvrement pour des allocations en ligne de flux.

En comparaison des recherches citées, les deux innovations principales de notre contribution sont :

- L'utilisation d'un routage hybride pour une tolérance aux pannes efficiente dans les réseaux temps réel maillés et notamment les réseaux de type SDRN ;
- La combinaison d'un mécanisme de routage en mode recouvrement et des méthodes d'analyse temporelle pour la caractérisation de la résilience des chemins durant le contrôle

d'admission. Cette caractérisation est agnostique du futur trafic et des sous-chemins de secours effectifs.

3.8 Conclusion

Ce chapitre étudie la problématique de la tolérance aux défaillances permanentes, ou tolérance aux pannes, dans les réseaux SDRN. Nous nous concentrons sur les pannes des éléments intermédiaires. La tolérance aux pannes des éléments terminaux est un problème classiquement séparé, puisqu'il apporte des problématiques supplémentaires liées aux couches applicatives. Notre objectif est de garantir le respect des contraintes temps réel des flux de communication, en sachant qu'un lien ou un nœud peut tomber en panne de façon définitive.

L'architecture SDRN est naturellement capable de réagir aux pannes par le biais du mécanisme de configuration en ligne. Lorsqu'une panne est détectée, le contrôleur en est informé puis reconfigure les flux impactés. Ce mécanisme valorise l'utilisation d'une topologie résiliente telle qu'une topologie maillée, qui propose plusieurs chemins possibles d'une source à une destination données.

Cependant, la période de transitoire de reconfiguration, autrement appelée période de recouvrement, est un intervalle d'incertitude pendant lequel les flux impactés par la panne peuvent ne pas tolérer une rupture temporaire de leurs communications. Notre problématique consiste donc à garantir à ces flux le respect des contraintes temps réel pendant la période transitoire de reconfiguration.

Cette période transitoire est étonnamment peu étudiée dans la littérature. En effet, une méthode habituellement utilisée consiste à dupliquer les flux de communication sur des chemins disjoints, ce qui garantit qu'au moins un des doublons arrivera à destination malgré l'apparition d'une panne sur un élément intermédiaire. Dans ce cas, il n'est pas nécessaire d'étudier la période transitoire. Cependant, cette méthode a l'inconvénient d'une sur-utilisation des ressources réseau, inutile pendant la majeure partie de la vie du réseau.

La contribution présentée dans ce chapitre, Hybrid Routing in SDRN (HYROS), consiste à combiner deux paradigmes de routage : un routage orienté donnée en mode nominal, optimisé et rigide, et un routage saut par saut en période de recouvrement, moins maîtrisé mais flexible. Cette méthode permet à des paquets critiques de continuer à arriver à destination à temps pendant la période de reconfiguration, sans nécessiter de calculer les chemins de secours pour chaque flux. En effet, les paquets en mode recouvrement utilisent des tables de routage statiques calculées à l'initialisation. L'utilisation de ce routage statique est possible grâce au changement de paradigme vers un routage orienté machine en mode recouvrement. De plus, ce routage statique, en saut par saut, apporte de bonnes propriétés de flexibilité.

Le Tableau 3.5 reprend le tableau des caractéristiques établi en Section 3.3.1, en y faisant apparaître le routage en mode nominal (en bleu) et le routage en mode recouvrement (en rouge). On observe que la combinaison des deux paradigmes nous permet de profiter des avantages sur les caractéristiques les plus importantes dans chaque mode, à savoir l'optimisation de l'utilisation du réseau en mode nominal, et la flexibilité et faible surcoût du contrôle d'admission en mode recouvrement.

Nous avons décrit la méthode d'utilisation d'HYROS, sous la forme d'une extension de l'architecture SDRN. Cette extension est notamment composée d'une méthode de caractérisation de la résilience d'un chemin, utilisée par le contrôle d'admission, ainsi que d'une utilisation originale du canal virtuel non temps réel. Ce dernier agit en effet comme un canal de secours partagé par les flux en mode recouvrement, mais utilisable par d'autres communications en mode nominal.

TABLE 3.5 – Caractéristiques des deux paradigmes composant HYROS (nominal en bleu, recouvrement en rouge)

	Statique ou dynamique	**Source ou saut par saut**	**Orienté machine ou donnée**
Surcoût*	adaptatif > **statique**	source > **saut par saut**	donnée > **machine**
Optimisation*	**adaptatif** > statique	**source** > saut par saut	**donnée** > machine
Adaptabilité*	adaptatif = statique	**saut par saut** > source	donnée = machine

Cette souplesse garantit une utilisation maximale des ressources réseau.

Une évaluation nous a permis de montrer l'intérêt de notre approche par rapport à deux approches naïves que sont l'approche par chemins disjoints et l'approche par sous-chemins. Globalement, la force du paradigme de routage utilisé en mode recouvrement est de pouvoir fonctionner comme une approche par sous-chemins, mais sans nécessiter le calcul des sous-chemins effectifs, grâce à l'utilisation des tables de routage statiques calculées à l'initialisation. Notre approche hybride permet de disposer de ces bonnes propriétés pour gérer les pannes, tout en conservant la bonne maîtrise du réseau proposée par l'architecture SDRN nominale.

La méthode que nous avons présentée est capable de tolérer un nombre infini de pannes, à condition qu'il n'y en ai pas plusieurs durant la même période de recouvrement. Il s'agit d'une hypothèse raisonnable, compte tenu de la capacité de l'architecture SDRN à reconfigurer rapidement les flux impactés et de la probabilité qu'une panne survienne. Cependant, une perspective naturelle de nos travaux est d'étudier la gestion des pannes lorsque plusieurs surviennent en même temps. Cette étude est notamment intéressante lorsque les pannes sont dues à des dégâts matériels : choc, tir d'arme, etc. Dans ces cas, la probabilité que plusieurs pannes surviennent en même temps peut devenir non négligeable.

Une autre perspective est de proposer une utilisation plus optimisée du réseau en mode recouvrement. Ce mode est utilisé parce qu'il permet de troquer des propriétés d'optimisation contre des propriétés de flexibilité et de faible surcoût, nous acceptons donc que l'utilisation du réseau soit moins maîtrisée en mode recouvrement. Cependant, réduire le pessimisme en mode recouvrement permettrait d'y accepter plus de flux. En effet, notre méthode d'utilisation du canal de recouvrement suppose que, dans le pire cas, les flux en mode recouvrement peuvent tous passer par le même lien. En ajoutant par exemple des mécanismes de répartition de charge, on pourrait réduire ce pessimisme et ainsi permettre à plus de flux d'accéder au mode recouvrement. Puisque les flux critiques sont dupliqués lorsque le mode recouvrement ne peut plus les accueillir, ceci améliorerait encore l'utilisation des ressources réseau dans le cas où de nombreux flux requièrent l'accès au mode recouvrement.

En résumé, HYROS est une méthode de routage hybride permettant une tolérance aux pannes efficiente dans un contexte temps réel, grâce à la combinaison de deux paradigmes de routage opposé. Cette combinaison a l'avantage d'optimiser l'utilisation du réseau en mode nominal, tout en permettant aux paquets de continuer à arriver à destination à temps pendant la période de reconfiguration. Grâce au basculement de paradigme, ceci est possible avec un faible surcoût. Cette contribution est la première, à notre connaissance, à combiner des mécanismes de routage hybride pour la tolérance aux pannes et des mécanismes d'analyse temporelle de flux temps réel, dans notre cas apportés par l'architecture SDRN.

Chapitre 4

Gestion des défaillances transitoires par approche probabiliste (résumé)

Sommaire

4.1 Introduction

La contribution proposée dans ce chapitre est une approche probabiliste pour la prise en compte des retransmissions dans le contrôle d'admission SDRN. Cette méthode permet une gestion sûre des erreurs de transmission, sans redondance (duplication) des communications. Ses nouveautés sont de forcer le respect de l'exigence de fiabilité au lieu de le vérifier *a posteriori*, ainsi que de permettre l'allocation incrémentale des ressources réseau sous contrainte de fiabilité. Elle fait l'objet d'un dépôt de brevet [Greff, 2018]. Pour cette raison, ce chapitre présente la problématique et l'état de l'art, puis résume l'idée de la contribution.

4.2 Problématique

Le Chapitre 2 présente l'architecture SDRN sous hypothèse de fiabilité totale des éléments du réseau. Le Chapitre 3 étudie la gestion des défaillances permanentes (pannes), à travers l'utilisation d'une solution de routage hybride. Dans ce chapitre, nous nous intéressons aux défaillances transitoires, c'est-à-dire aux erreurs de transmission. Un paquet réseau peut être corrompu à cause d'une erreur matérielle conduisant à une inversion de bit. Ces erreurs sont des événements très rares mais doivent être gérées car elles altèrent l'intégrité du paquet. Classiquement, l'intégrité des paquets est vérifiée via des codes détecteurs d'erreurs ou sommes de contrôle ajoutés en queue. Si le récepteur détecte une erreur qu'il ne sait pas corriger, le paquet est détruit et une notification est éventuellement retournée à l'émetteur.

Une solution très classique pour la gestion de ces erreurs dans un contexte temps réel est de dupliquer les communications le long de chemins disjoints (Figure 4.1). Cette solution prend l'hypothèse que les erreurs de transmission sont si rares qu'une donnée en subira au plus une sur

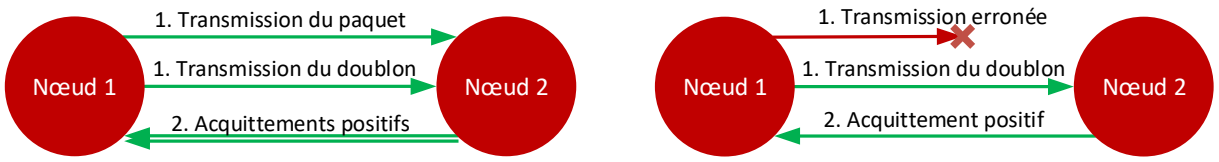

FIGURE 4.1 – Mécanisme de duplication

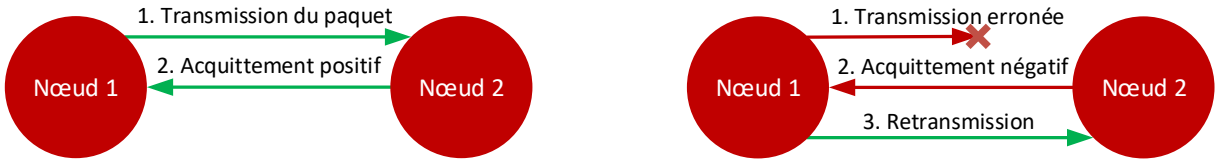

FIGURE 4.2 – Mécanisme de retransmission

l'un des deux chemins. Si cette erreur fait que la donnée arrivera trop tard, alors au moins son doublon arrivera à destination à temps. Cette solution possède deux limites. Premièrement, elle double les ressources réseau utilisées. Ce surcoût est inutile durant la majeure partie de la vie du réseau, car les erreurs de transmission sont rares. Deuxièmement, elle suppose qu'au moins un doublon arrivera à destination sans erreur, ce qui a une probabilité de ne pas être vrai. Ce risque est calculé *a posteriori*, comparé à une exigence statistique connue par le concepteur du système, puis le nombre de doublons utilisés est éventuellement augmenté.

Nous étudions l'utilisation des mécanismes de retransmission pour gérer les erreurs de transmission sans redondance des communications. Nous nous intéressons donc aux réseaux capables de retransmettre un paquet ayant subi une erreur de transmission. Il s'agit d'un mécanisme très classique mais pas toujours présent. Ce mécanisme est le suivant : si la réception d'un paquet n'a pas été acquittée par le voisin après un temps d'attente, ou que ce dernier a envoyé un acquittement négatif, alors l'émetteur retransmet le paquet (Figure 4.2). Cela permet de s'assurer qu'une donnée arrivera à destination, même si elle a subi des erreurs de transmission. De plus, les ressources réseau supplémentaires ne sont utilisées qu'en cas d'erreur, ce qui constitue un avantage important par rapport aux approches par redondance.

Dans un système temps réel, on a néanmoins la problématique suivante. La retransmission d'un paquet signifie que l'on a envoyé deux paquets au lieu d'un, ce qui retarde l'arrivée de la donnée complète et peut lui faire manquer son échéance. Il existe également une probabilité, plus faible, qu'un train de paquets correspondant à une donnée subisse deux erreurs de transmission, et ainsi de suite. Le nombre de retransmissions pouvant potentiellement être infini, nous devons donc utiliser une approche probabiliste pour garantir l'arrivée des données malgré les retransmissions.

4.3 Approches probabilistes dans les systèmes temps réel

Il existe des approches probabilistes utilisées dans l'analyse temporelle des systèmes temps réel [Quinton et al., 2012]. Elles permettent notamment de répondre aux problématiques de certification imposant des contraintes statistiques, notamment dans le domaine avionique [Scharbarg et al., 2009]. Elles sont également utilisées pour réduire l'ensemble des connaissances nécessaires à l'analyse temporelle [Cucu-Grosjean et al., 2012] ou construire des approches optimistes d'analyse temporelle. Enfin, on pourra citer leur utilisation pour l'analyse de tâches dont certains paramètres varient selon une loi statistique [Maxim and Cucu-Grosjean, 2013].

L'exigence statistique à laquelle on veut obéir est classiquement représentée par un paramètre, la **fiabilité** (*reliability* en anglais). Il s'agit de la probabilité qu'aucune défaillance non gérée n'ait lieu durant l'intervalle de temps considéré [Kopetz, 1997]. Elle est équivalente à un temps moyen entre défaillances transitoires, que l'on obtient en divisant l'intervalle de temps considéré par la fiabilité :

$$temps\ moyen\ entre\ défaillances = \frac{intervalle\ considéré}{(1 - fiabilité)} \tag{4.1}$$

Dans notre contexte d'étude des retransmissions, une défaillance correspond au cas où une donnée subit plus de retransmissions que la borne que l'on a fixée.

Dans [Burns et al., 1999], un modèle d'erreurs basé sur un processus (et une loi statistique) de Poisson est dérivé pour obtenir la probabilité que, sur la durée de vie du système, les tâches ne vont pas subir plus de retransmissions que prévu. Cet article utilise la notion de temps moyen entre défaillances transitoires, ou *Mean Time Between transient Faults (MTBF)*. En d'autres termes, elle explore la notion de garantie d'ordonnançabilité probabiliste, dont l'expression est le temps moyen séparant deux instants où le système n'est pas ordonnançable. [Sebastian and Ernst, 2009] et [Sebastian and Ernst, 2008] raffinent ce mécanisme avec une approche par tâche ou communication, dont on combine ensuite les paramètres statistiques pour obtenir la fiabilité globale. Cette approche réduit la complexité de l'analyse.

Dans le domaine des réseaux automobiles (CAN notamment), les travaux de [Wilwert et al., 2003] puis [Simonot et al., 2005] étudient l'influence sur la fiabilité du système des erreurs de communication dues aux interférences électromagnétiques. Les défaillances transitoires peuvent avoir lieu jusqu'à un certain seuil, au-delà duquel l'application et le système sont défaillants. Les travaux utilisent un modèle probabiliste des interférences électromagnétiques pour en déduire une probabilité que le système soit défaillant. Cette analyse tient compte du fait qu'un certain nombre de défaillances consécutives peuvent être tolérées. Enfin, Navet [Navet et al., 2000] introduit la notion de *worst-case deadline failure probability*. L'approche est similaire aux précédentes à ceci près qu'elles contient la notion de recouvrement d'erreurs, c'est-à-dire qu'un message peut arriver à temps à destination malgré les erreurs de transmission. Là encore, un seuil d'erreurs de transmissions au-delà duquel l'échéance est manquée est défini et une probabilité de le dépasser est calculée.

Ces méthodes permettent de prendre en compte les retransmissions dans l'analyse temporelle. Elles ont néanmoins deux limites, auxquelles notre contribution répond :

— La fiabilité du système est calculée *a posteriori* puis comparée à l'exigence. Si elle est plus faible, alors il faut itérer le processus de conception. En d'autres termes, la fiabilité du système est une conséquence, elle est observée. Notre contribution utilise l'exigence de fiabilité comme paramètre d'entrée et alloue les ressources réseau de sorte que la fiabilité résultante obéisse à l'exigence ;

— La fiabilité globale prend en compte l'ensemble des tâches, ce qui empêche une analyse incrémentale, essentielle à l'application du SDRN. Notre contribution décompose l'exigence de fiabilité pour permettre son utilisation de façon incrémentale.

4.4 Résumé de la contribution

La solution proposée est une extension du contrôle d'admission SDRN garantissant une gestion sûre des erreurs de transmissions selon une contrainte de fiabilité, le tout sans redondance des communications.

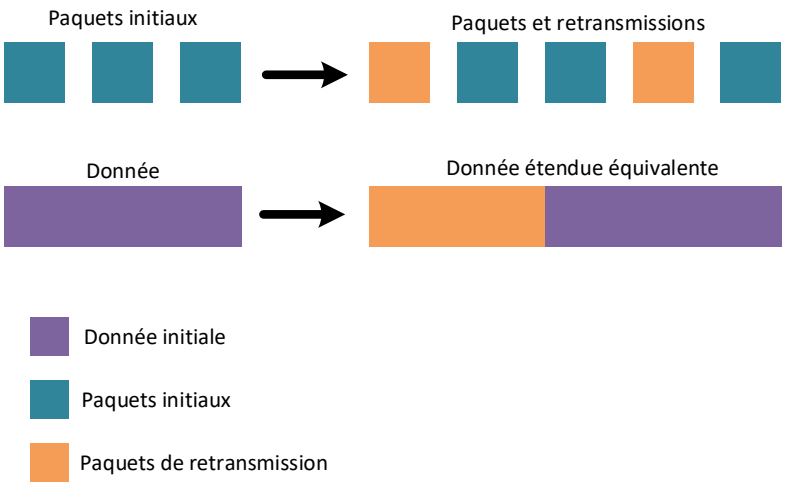

FIGURE 4.3 – Extension de la taille de donnée à cause des retransmissions

Le principe général est de borner le nombre de paquets de retransmission que peut subir une donnée, afin d'en déduire une taille de donnée étendue équivalente dans le pire cas (Figure 4.3). Cette taille de donnée est prise en compte par le contrôleur SDRN à la place de la taille de donnée précisée dans la requête de flux. Au final, on alloue les mêmes ressources que si la donnée était par défaut plus grande du nombre de paquets de retransmission calculés. Bien entendu, il existe une probabilité que le nombre de retransmissions subies par la donnée dépasse cette borne, ce qui correspond à une défaillance. Notre méthode d'analyse est telle que la fiabilité totale du réseau sera supérieure ou égale à l'exigence de fiabilité donnée en entrée.

La méthode est décomposée en trois parties :

1. Décomposer l'exigence de fiabilité en exigences individuelles par flux, ce qui permet l'allocation incrémentale ;
2. Lors du contrôle d'admission, analyser le pire cas en termes de retransmissions, tel que la fiabilité du flux est supérieure ou égale à l'exigence ;
3. Remplacer la taille de donnée par sa taille étendue équivalente pire cas dans le contrôle d'admission.

Cette méthode se positionnant comme une extension du contrôle d'admission, elle s'intègre facilement à l'architecture SDRN et apporte un faible surcoût au temps de contrôle d'admission.

Chapitre 5

ERICA, méthode d'expérimentation hybride de réseaux

Sommaire

5.1 Introduction

L'expérimentation est un processus essentiel dans la conception de systèmes cyber-physiques. Elle s'effectue classiquement par le biais de la simulation ou de l'utilisation de plateformes d'expérimentation physiques.

Comme nous l'avons indiqué en avant-propos, la réalisation d'une plateforme matérielle de démonstration sur RapidIO fait partie de notre cahier des charges. Cependant, la réalisation d'une plateforme matérielle possédant beaucoup d'inertie, il est nécessaire de définir la base de l'architecture par un autre biais. Ceci nous a orienté vers l'utilisation de la simulation comme moyen de conception initial. Puisqu'il n'existe pas, à notre connaissance, de simulateur de réseau RapidIO ouvert, il était nécessaire de le concevoir.

La première contrainte de la simulation est d'être une représentation suffisamment pertinente de la plateforme réelle, de sorte que les solutions conçues soient cohérentes avec les systèmes réels. De plus, l'utilisation, dans la simulation, d'applications correspondant aux applications métier destinées à utiliser le réseau, bonifie l'expérimentation. Par exemple, nous souhaitons que la couche applicative soit sensible aux problématiques du temps réel et de la performance du système. Ceci nous a interrogé sur l'utilisation d'applications réelles dans un contexte d'expérimentation réseau. Enfin, les expérimentations sont destinées à tester différentes solutions ou scénarios. La méthode avec laquelle ces derniers sont mis en place et modifiés a donc un impact significatif sur l'efficacité du processus d'expérimentation.

L'ensemble de ces considérations nous a amené à porter une réflexion sur notre méthode d'expérimentation, ce qui a mis en lumière des défis scientifiques intéressants qui seront abordés dans ce chapitre. En conséquence, nous avons pensé une méthode d'expérimentation originale, ERICA (pour ExpeRImenting Cross-layer quality of service for Autonomic systems). Cette méthode est dédiée à l'expérimentation hybride de réseaux, c'est-à-dire l'expérimentation utilisant des parties réelles et simulées dans le même processus. Son avantage est de faciliter la construction et la modification de piles d'expérimentations hybrides. On appelle pile d'expérimentation l'ensemble des couches de l'expérimentation, dans notre cas le réseau simulé ou réel et les applications communication par-dessus. On parle de pile d'expérimentation hybride lorsqu'elle mélange des éléments ne provenant pas du même environnement, par exemple des applications réelles et un réseau simulé. La méthode que nous proposons aborde également la problématique de l'utilisation transparente d'applications réelles sur un réseau simulé, via un mécanisme d'hybridation générique et un mécanisme de synchronisation temporelle.

Ce chapitre décrit la méthode d'expérimentation ERICA, sa mise en œuvre dans le cadre de notre étude, les défis scientifiques auxquels nous avons proposé une contribution originale, puis sa générisation sous la forme d'un *framework* d'expérimentation.

Dans un premier temps, nous allons positionner cette contribution par rapport à la littérature. Puis nous décrirons les concepts principaux d'ERICA. Ensuite, nous détaillerons sa mise en œuvre et la façon dont nous avons résolu les problèmes scientifiques les plus importants. Enfin, nous présenterons la générisation de cette méthode sous la forme d'un logiciel évolutif, dédié à la mise en place d'expérimentations hybrides de réseaux.

5.2 Positionnement

Le principe d'utiliser un simulateur pour émuler une partie d'un système plus complexe a été mis en lumière par le domaine de la simulation parallèle et distribuée (*Parallel And Distributed Simulation*, PADS)[Fujimoto, 2001]. La PADS aborde le besoin de distribuer les éléments d'une simulation complète, dans le but de gagner en rapidité de simulation par la parallélisation des tâches, de réutiliser des composants de simulation existants ou d'utiliser des composants physiquement distribués dans un environnement (par exemple en simulation militaire pour l'infanterie).

High Level Architecture (HLA) [IEEE, 2010] est le standard phare pour le couplage de

composants hétérogènes dans une architecture d'expérimentation plus globale [Perumalla, 2006] [Strassburger et al., 2008]. Il répond au besoin de coupler des simulateurs et/ou utilisateurs réels interagissant en temps réel (*live players*) sans avoir à concevoir une architecture d'expérimentation de zéro [Dahmann and Morse, 1998]. L'ensemble des éléments collaborant est appelée une fédération. HLA propose des interfaces de couplage de composants matériels avec l'expérimentation (*Hardware-in-the-Loop*). Il spécifie également une méthode de synchronisation temporelle entre composants parfois hétérogènes en termes de gestion du temps [Perumalla, 2006] [Carothers et al., 1997]. Des extensions pour le simulateur Ptolemy II, permettant son intégration dans une fédération HLA, ont été conçus [Lasnier et al., 2013]. Cependant, HLA s'avère être généralement très long et complexe à utiliser pour concevoir une architecture complète [Strassburger et al., 2008]. De plus, il n'existe pas de méthode standard pour son utilisation, menant au besoin de paquets additionnels pour l'interopérabilité [Taylor, 2003] [Taylor et al., 2006].

MECSYCO [Camus et al., 2015] combine le formalisme Discrete EVent System Specification (DEVS) [Zeigler et al., 2000] et le paradigme multi-agents pour gérer l'interopérabilité des formalismes et simulateurs. Il est notamment utilisé pour la simulation de *Smart Grids* par composition de simulateurs existants [Vaubourg et al., 2015]. De plus, il ne permet pas de combiner des simulateurs avec des éléments réels, ce qui est nécessaire pour pouvoir construire des piles d'expérimentation hybrides. Étendre MECSYCO avec la fonctionnalité d'hybridation est une perspective intéressante. Ayant besoin d'une solution rapidement fonctionnelle, nous avons préféré construire une première solution indépendante, qui pourra être intégrée plus tard à MECSYCO.

COOJA [Osterlind et al., 2006] est un simulateur de réseaux de capteurs sans fil. Il inclut un émulateur de micro-contrôleurs (par exemple MPSim pour émuler un MSP430 MPU) et un simulateur de canal radio simple. Cela permet au code source et système d'exploitation du capteur de s'exécuter soit dans COOJA soit sur un nœud capteur. Bien que cette approche s'applique à un domaine de recherche différent du nôtre, elle est très proche dans le principe d'utiliser différents environnements d'expérimentation au sein du même processus de conception.

La simulation symbiotique est un paradigme dans lequel une simulation et un système cyber-physique sont étroitement associés [Aydt et al., 2009]. Ce concept est essentiellement utilisé pour de l'analyse de situation dite *what-if analysis*. Cela consiste à contrôler le système en temps réel via une simulation permettant d'analyser différents scénarios complexes. Ce paradigme met également en valeur l'idée d'utiliser des données provenant du système réel (par exemple données de capteurs) pour améliorer la simulation. Comme nous le détaillerons plus tard, ce principe est essentiel à notre approche. En effet, nous proposons d'utiliser des données capturées du système réel pour améliorer la précision de la simulation et la régler au fur et à mesure que l'on expérimente les solutions conçues, ce que nous appelons la symbiose.

Enfin, les projets Emulab [White et al., 2002], Common Open Research Emulator (CORE) [Ahrenholz et al., 2008] et cOntrol, Management and Measurement Framework (OMF) [Rakotoarivelo et al., 2010] sont des plateformes d'expérimentation de réseaux proposant d'hybrider dans le même processus d'expérimentation les trois notions fondamentales que sont la simulation, l'émulation et le banc de test physique. Les environnements d'expérimentation intégrés à ces projets sont néanmoins limités. Network Experimentation Programming Interface (NEPI) [Quereilhac et al., 2011] [Lacage et al., 2009] propose de faire évoluer ce concept via une interface de programmation d'expérimentations ouverte en termes d'environnement d'expérimentation. Ceci permet à l'utilisateur de faire fonctionner l'expérimentation dans le ou les environnements les plus pertinents, afin d'évaluer et reproduire les mêmes scénarios d'expérimentation, offrant ainsi une meilleure fiabilité en termes de résultats. La méthode ERICA que nous décrivons dans ce chapitre se base sur un principe similaire : utiliser plusieurs environnements d'expérimenta-

tion dans le même processus de conception et permettre au chercheur de construire des piles d'expérimentations hybrides composées de couches arbitraires, au sein d'une interface graphique homogène.

5.3 Concepts principaux

5.3.1 Expérimentation multi-plateformes

On appelle plateforme d'expérimentation, ou environnement d'expérimentation, un simulateur, logiciel d'émulation ou banc de test matériel utilisé pour la conduite d'une expérimentation réseau. Lorsque le processus d'expérimentation fait intervenir une unique plateforme, par exemple un simulateur, on parle d'expérimentation mono-plateforme. Lorsque deux ou plusieurs plateformes sont utilisées indépendamment pour observer le déroulement du même scénario, on parle d'expérimentation multi-plateformes. Par exemple, on utilise deux simulateurs différents pour valider un modèle de simulation. Enfin, lorsque deux ou plusieurs plateformes sont utilisées au cœur d'une même expérimentation, on parle d'expérimentation cross-plateformes ou hybride. Il peut s'agir de coupler des éléments simulés avec des éléments réels, ou encore d'effectuer une co-simulation.

L'un de nos objectifs consiste en l'étude et la démonstration de la faisabilité industrielle des réseaux SDRN ; plus particulièrement, la réalisation d'un démonstrateur sur RapidIO. Il est donc indispensable qu'une plateforme matérielle soit présente au sein de notre méthode d'expérimentation. Cependant, une telle plateforme apporte certains inconvénients rendant son utilisation exclusive peu pertinente. Premièrement, son développement est long et complexe. Une modification fréquente afin de tester différentes solutions est donc inefficace. En conséquence, l'utilisation de l'expérimentation pour faire apparaître des scénarios critiques lors des phases d'immaturité du protocole est limitée, ce qui oblige à une vérification théorique plus poussée en amont. Deuxièmement, il n'est pas possible de s'abstraire de certaines parties du matériel afin de simplifier ou isoler l'observation d'un mécanisme particulier. Les incertitudes de fonctionnement du matériel (par exemple, latence d'accès mémoire) complexifient également l'observation. Troisièmement, l'observation du passage à l'échelle est limitée par les moyens matériels disponibles, tandis que des réseaux beaucoup plus grands et de caractéristiques différentes peuvent être construits par le biais de la simulation. Pour l'ensemble de ces raisons, nous incluons l'utilisation de la simulation à notre processus d'expérimentation. Celle-ci doit permettre de disposer d'un environnement flexible pour la conception et le test de solutions réseau.

Le premier concept de la méthode ERICA est donc l'utilisation de plusieurs plateformes ou environnements dans le processus d'expérimentation.

5.3.2 Hybridation et symbiose

Le simulateur que nous souhaitons mettre en place doit répondre à plusieurs critères. D'une part, il doit proposer une grande flexibilité au niveau des mécanismes clés du protocole : ordonnancement des paquets, routage, etc. Il doit également être suffisamment simple pour être développé et évoluer en un temps raisonnable. D'autre part, il doit constituer une représentation suffisamment précise de la plateforme matérielle, afin que l'évaluation des solutions simulées soit pertinente. Nous proposons deux concepts pour répondre à ces critères :

— **Hybridation** : la partie haute des expérimentations (système d'exploitation, applications, intergiciel éventuel, etc.) est constituée d'éléments réels couplés avec et communicant via le réseau simulé. Ceci présente deux avantages. Premièrement, nous nous affranchissons

du besoin de simuler précisément ces éléments, tout en ayant un comportement parfaitement réaliste (car réel). Le réseau est la seule partie nécessitant la flexibilité et simplicité évoquées plus haut, il est donc acceptable de ne pas maîtriser le détail des couches hautes. Deuxièmement, cela permet d'utiliser la même couche supérieure sur le réseau simulé que sur la plateforme matérielle, ce qui nous assure que les différences constatées entre les deux environnements ne proviennent que du réseau. Dans le cas où les applications seraient simulées, une partie des différences constatées pourraient provenir d'un manque de précision ou d'une erreur dans le modèle de simulation des applications. Enfin, cela donne la possibilité d'utiliser de vraies applications métier, ce qui a un intérêt à la fois en termes d'expérimentation mais également de démonstration. Cette approche est similaire aux approches de *Hardware-in-the-Loop*, à ceci près que, dans notre cas, c'est la couche applicative qui est constituée d'éléments réels couplés au reste de l'expérimentation.

— **Symbiose** : la plateforme matérielle est utilisée pour régler et valider le modèle de simulation, garantissant sa pertinence tout en permettant une construction plus simple. En retour, le simulateur permet de concevoir des solutions qui seront ensuite implémentées sur la plateforme matérielle. Cette interaction mutuellement enrichissante est la définition de la symbiose. Pour cette raison, nous définissons notre approche comme une approche hybride symbiotique, résumée en Figure 5.1.

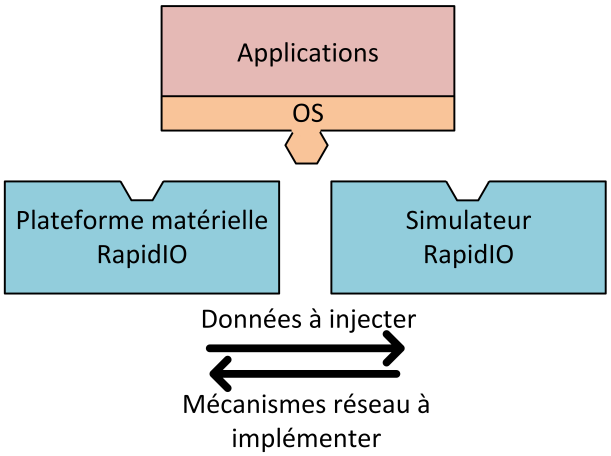

FIGURE 5.1 – Mise en œuvre d'ERICA : hybridation et symbiose

5.3.3 Automatisation

La mise en place des scénarios d'expérimentation, le démarrage de la plateforme ou de la simulation, le déploiement et le lancement des applications, sont autant d'actions ayant un impact non négligeable sur l'efficacité du processus d'expérimentation. Cette observation est encore plus importante lorsqu'on s'inscrit dans une approche symbiotique telle que décrit ci-dessus, avec une modification régulière d'une partie de la pile d'expérimentation.

ERICA répond à cette problématique en proposant une automatisation de la mise en place des expérimentations. L'utilisateur définit, via une interface graphique par exemple, l'expérimentation qu'il souhaite mettre en place, puis l'ensemble des fichiers et scripts nécessaires à l'exécution de l'expérimentation sont générés. En plus de faciliter la mise en place, cette méthode a l'avantage de découpler la connaissance de l'environnement d'expérimentation de la connaissance de

l'expérimentation à mener. Par exemple, il n'est plus nécessaire de connaître en détail le simulateur utilisé. Une fois que la méthode d'automatisation a été implémentée, les détails techniques bas niveaux sont masqués de l'utilisateur. De cette façon, le chercheur peut mener une réflexion plus haut niveau et fonctionnelle sur ses expérimentations.

Enfin, le fait de représenter la pile d'expérimentation à l'extérieur de l'environnement d'expérimentation utilisé découple ce dernier du scénario. En effet, le scénario spécifié (par exemple, exécuter une application A sur un réseau RapidIO de quatre nœuds) est indépendant de l'environnement d'expérimentation, qui devient un paramètre de la génération. Par exemple, mettre en place l'expérimentation définie dans un simulateur choisi, ou la déployer sur une plateforme choisie. Ceci rend le changement d'environnement d'expérimentation très simple, ce qui facilite la mise en œuvre du mécanisme de symbiose décrit dans la partie précédente.

5.3.4 Architecture générale

L'architecture d'expérimentation illustrée en Figure 5.1 est générisée en incluant les concepts décrits dans les parties précédentes. Cette architecture générique d'ERICA est représentée en Figure 5.2. Nous décrirons dans la section suivante comment nous la mettons en œuvre en pratique.

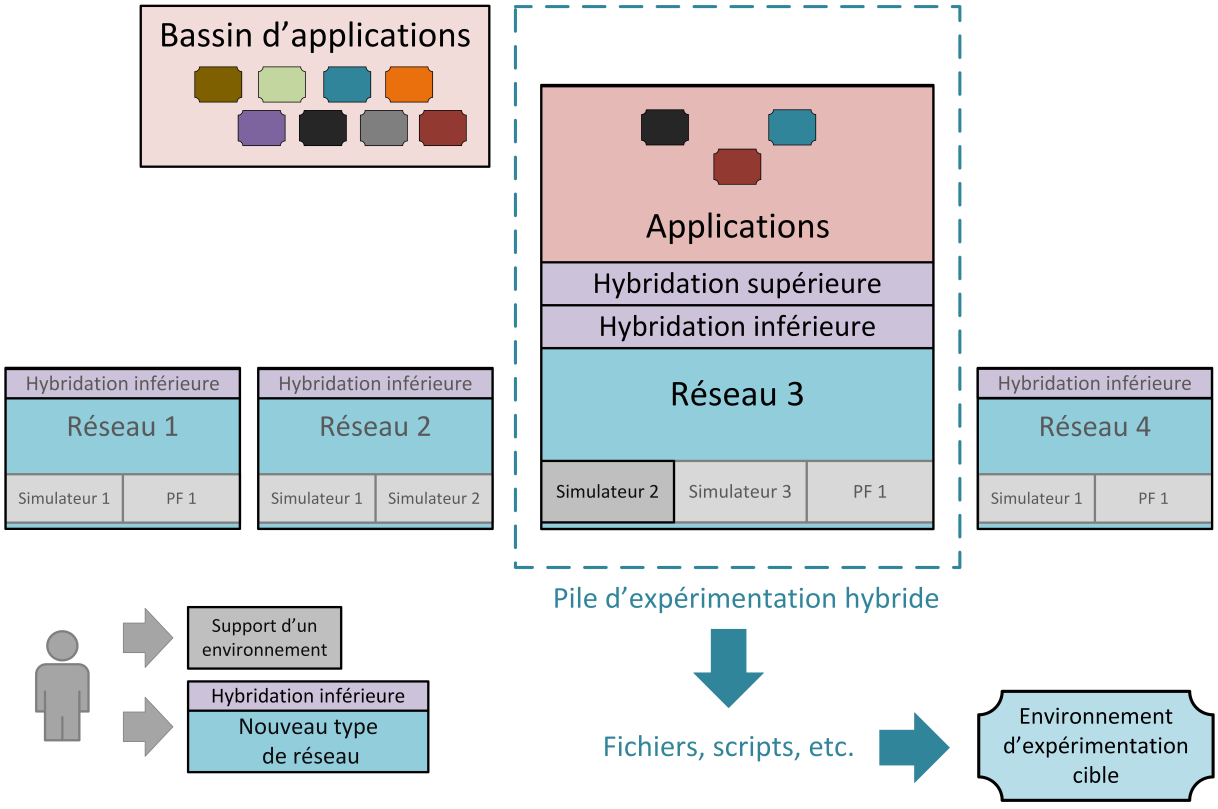

FIGURE 5.2 – Architecture générale d'ERICA

Le principe d'ERICA est de construire une pile d'expérimentation, composée d'applications s'exécutant sur un réseau selon un scénario défini. Ce réseau peut être simulé ou réel, c'est pourquoi la pile d'expérimentation est dite hybride. ERICA génère ensuite les éléments nécessaires pour la mise en œuvre automatique de l'expérimentation définie dans l'environnement cible.

Le réseau, le scénario et l'environnement d'expérimentation peuvent être remplacés. Lorsque seul l'environnement d'expérimentation change, cela se fait de façon transparente du point de vue des applications. Seul le résultat de la génération est impacté. Le processus d'expérimentation est ainsi approché de façon inversée par rapport aux paradigmes classiques (Figure 5.3). En effet, la pile d'expérimentation est classiquement construite en partie ou totalité au sein de l'environnement d'expérimentation, ce qui limite la réutilisabilité des éléments qui la composent : par exemple, description du scénario, hybridation, etc.

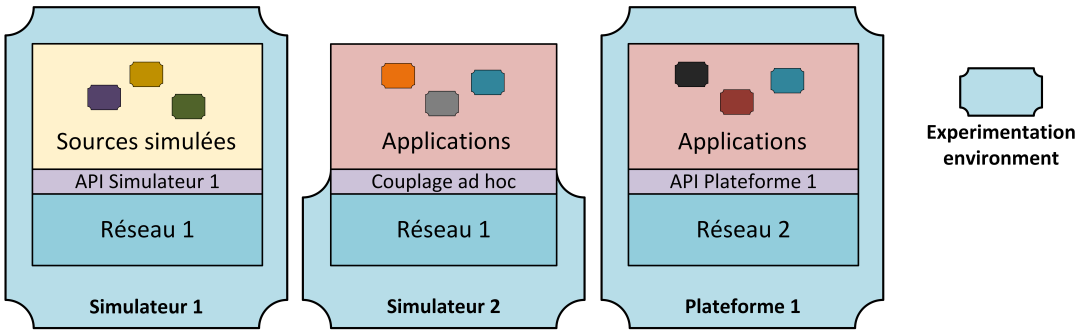

FIGURE 5.3 – Représentation des paradigmes d'expérimentation classiques

Pour chaque partie basse (type de réseau et environnement), il est bien entendu nécessaire de définir une fois la façon dont l'automatisation de la mise en place de l'expérimentation doit s'effectuer (génération des fichiers nécessaires par exemple). Une fois ce travail réalisé, l'automatisation de la mise en place de l'expérimentation est transparente du point de vue de l'utilisateur. Puisque nous imaginons ERICA comme une architecture évolutive, il est nécessaire qu'un tiers puisse l'étendre avec le support d'un nouvel environnement ou d'un nouveau type de réseau.

La section suivante détaille la mise en œuvre de la méthode ERICA dans le cadre de l'expérimentation des réseaux SDRN sur RapidIO, tel qu'illustré en Figure 5.1.

5.4 Mise en œuvre : plateforme et simulateur RapidIO

Dans notre contexte, la mise en œuvre d'ERICA est constituée d'une plateforme matérielle et d'un simulateur, basés sur le protocole RapidIO (cf. Chapitre 1). Elle met en œuvre les mécanismes d'hybridation et de symbiose. L'objectif est de disposer d'un simulateur représentatif de la plateforme réseau pour servir de base d'expérimentation des mécanismes du SDRN. Nous ne nous intéressons pas à l'implémentation exacte de la plateforme et du simulateur, mais à l'application de la méthode ERICA et la résolution des défis scientifiques associés.

5.4.1 Architecture de la simulation

Notre objectif est d'utiliser la plateforme matérielle et le simulateur dans le même processus, à des fins de comparaison et d'enrichissement mutuel. Pour cette raison, nous concevons une architecture de simulation pertinente par rapport à l'architecture réelle de la plateforme matérielle.

Chaque nœud de la plateforme est composé d'une carte de type FPGA contenant l'implémentation matérielle du commutateur RapidIO, ainsi que d'un processeur exécutant les applications. Chacun possède quatre ports externes permettant de relier les nœuds entre eux via des liens RapidIO, et un port interne reliant le commutateur au processeur via un bus AXI. Les ports

communiquent tous en *full-duplex*. De plus, le passage du bus AXI à la partie RapidIO, et vice versa, se fait par un pont. Cette architecture matérielle est résumée en Figure 5.4. Habituellement, la logique interne des commutateurs et adaptateurs réseaux, lorsqu'ils sont pris sur étagère, n'est pas connue précisément. Dans notre cas, grâce à la disponibilité du code VHDL de l'implémentation, nous sommes capables de comprendre et analyser les comportements matériels en détail afin de corriger le modèle de simulation si nécessaire. De plus, l'outil de conception permet l'insertion de sondes dans le matériel afin de mesurer des valeurs précises de latences de communication. Ceci sera nécessaire à l'application de l'approche symbiotique, c'est-à-dire l'enrichissement mutuel entre la plateforme et le simulateur.

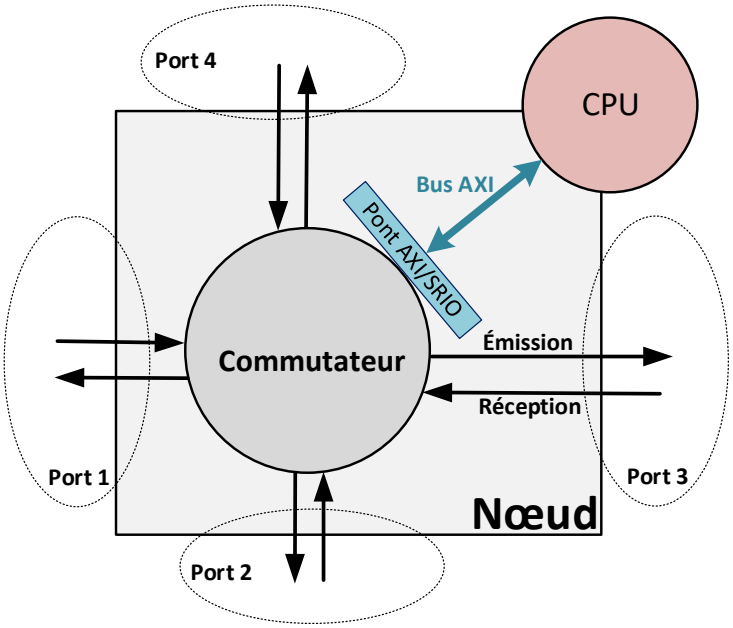

FIGURE 5.4 – Architecture matérielle simplifiée des nœuds de la plateforme SRIO

La simulation proposée se base sur un modèle à événements discrets de cette architecture générale. L'utilisation de modèles à événements discrets a été montrée comme appropriée à la simulation de réseaux distribués [Kurose and Mouftah, 1988] [Mouftah and Sturgeon, 1990]. Dans notre cas, nous utilisons le *framework* de modélisation Ptolemy II [UC Berkeley, 2014], développé à l'Université de Californie Berkeley. Ce logiciel permet de combiner modélisation graphique basée sur des acteurs partageant des entrées/sorties et modélisation basée sur des scripts.

5.4.2 Sections clé

Plutôt que de modéliser de façon très précise le fonctionnement interne du matériel (files d'attentes, tampons, logique, etc.), auquel nous n'allons apporter aucune modification ultérieure dans le simulateur, nous n'en conservons que les sections les plus significatives :

— La couche supérieure : processeur, mémoire, applications. Puisque nous allons utiliser l'hybridation pour la mettre en œuvre, elle est représentée dans le modèle mais non détaillée ;
— Le bus AXI et le pont AXI/SRIO ;
— Le commutateur réseau, constitué :

— Du commutateur en lui-même, orientant les paquets depuis un port d'entrée vers un port de sortie,
— De ports réseau *full-duplex*, constitués d'un sous-acteur d'émission et d'un sous-acteur de réception.

La simulation se concentre sur les fonctions clé du protocole, c'est-à-dire qui ont un impact significatif sur le déterminisme, le comportement des flux de communication, la commutation et les latences de communication. L'objectif est de posséder une représentation suffisamment pertinente de la réalité sans qu'elle soit parfaitement précise. Comme cela sera détaillé en Section 5.4.6, les mesures obtenues depuis la plateforme nous permettrons d'évaluer la précision du simulateur.

De même, les latences de passage dans chaque partie du matériel ne sont pas représentées individuellement, mais sont agrégées sous la forme de **sections clés** possédant une **latence globale**. Une section clé correspond à un enchaînement d'actions matérielles validant ces deux critères :

— Aucun embranchement n'est présent au sein de la chaîne, c'est-à-dire qu'une donnée entrant dans la section clé parcourra toujours les même sous-sections matérielles. A titre d'exemple, le passage du processeur au port de sortie ne peut être considéré comme une section clé, car la donnée est aiguillée par le commutateur entre les deux. En revanche, la traversée du port de sortie peut être considérée comme une section clé ;
— La latence globale d'un bout de la section à l'autre n'est pas négligeable.

Selon ce principe, les sections clés identifiées sont les suivantes (Figure 5.5) :

— Mémoire vers pont AXI/SRIO (lecture AXI) ;
— Traversée du pont AXI/SRIO ;
— Commutation ;
— Commutateur vers tampons d'émission ;
— Tampons d'émission vers ligne et propagation du paquet ;
— Ligne de réception vers tampon de réception ;
— Tampon de réception vers commutateur ;
— Pont AXI/SRIO vers mémoire (écriture AXI).

5.4.3 Scripts libres

Des scripts en Python sont utilisés pour modéliser les parties qui seront potentiellement fortement modifiées : routage et ordonnancement. L'utilisation de scripts offre un environnement de test souple et ouvert, permettant une grande expressivité. La composition hiérarchique de l'architecture de simulation est résumée en Figure 5.5.

5.4.4 Couche d'hybridation

Le second objectif dans la mise en place de la simulation est d'hybrider la partie réseau avec les couches logicielles. L'on souhaite que le système d'exploitation, applications et intergiciels soient réels et communiquent à travers le réseau simulé. Afin de réaliser cette interaction, nous devons faire face à deux défis. Le premier consiste à créer ce pont entre les deux couches, afin que les messages envoyés par les applications soient rejoués au niveau du réseau, et vice versa. Le deuxième consiste à synchroniser les vitesses d'exécution de la partie réelle et de la partie simulée, de sorte que les latences perçues par les applications correspondent aux latences simulées. Cette section se concentre sur le premier défi ; le second sera traité en Section 5.4.5.

Dans notre cas, la couche d'hybridation doit être capable de :

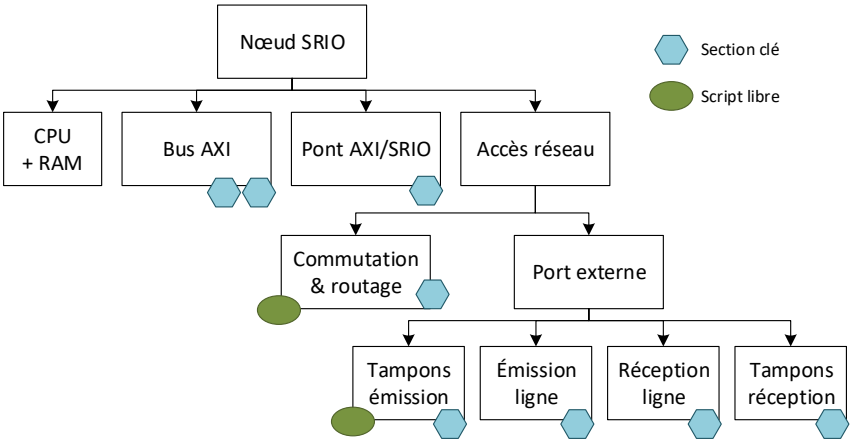

FIGURE 5.5 – Composition hiérarchique du modèle de simulation

— Émuler l'envoi d'une donnée depuis la section « processeur » du nœud simulé sur lequel on considère que l'application s'exécute ;
— Construire l'entête protocolaire simulé en fonction de l'application destinataire ;
— Transmettre les paquets reçus au niveau du nœud simulé à l'application réelle correspondante ;
— Proposer une configuration centralisée. On souhaite en effet pouvoir rapidement modifier la localisation des applications via une vue globale du mappage.

Pour ce faire, la couche d'hybridation est composée de deux parties (Figure 5.6). Le module supérieur modifie le comportement des applications pour que leurs communications soient dirigées vers l'environnement d'expérimentation, ici le simulateur. Le module inférieur est lié à la partie basse de la pile d'expérimentation. Dans notre cas, il est composé d'un ensemble d'extensions Ptolemy II.

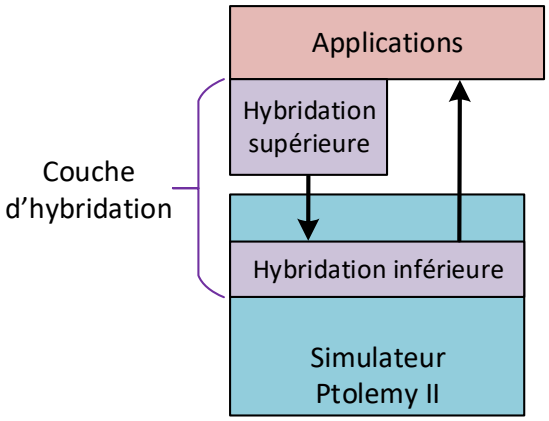

FIGURE 5.6 – Couche d'hybridation

Module d'hybridation supérieur

Le module supérieur est chargé d'intercepter les paquets envoyés par les applications et les transmettre à la couche d'hybridation inférieure, constituée des extensions Ptolemy II. Lorsqu'une application est sur le point d'envoyer un paquet par son interface réseau, le module d'hybridation

supérieur l'intercepte, l'encapsule et l'aiguille vers une socket d'écoute dans Ptolemy II. De cette façon, on va créer un tunnel Ptolemy II dans lequel va circuler le trafic IP original (Figure 5.7).

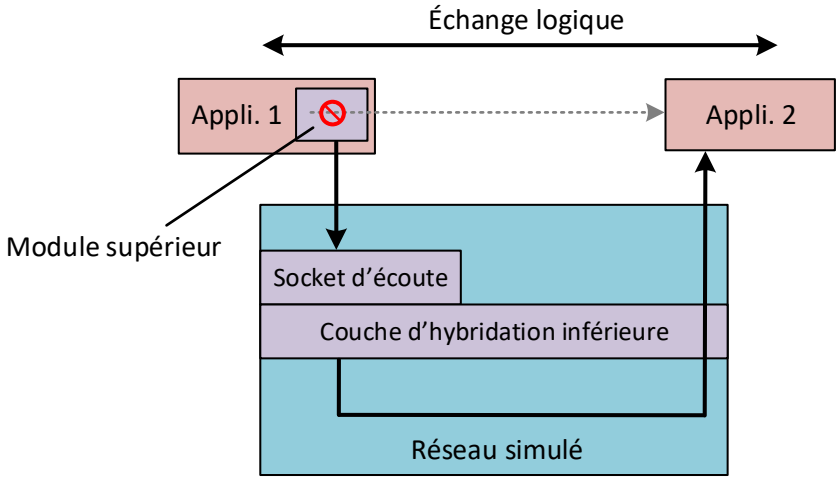

FIGURE 5.7 – Couche d'hybridation supérieure

Ce module peut être implémenté de différentes façons. L'une d'entre elles est de modifier le code des applications de sorte à opérer cette encapsulation au niveau de l'envoi dans la *socket* UDP/IP. Cette méthode a l'avantage d'être générique et applicable à toute application UDP/IP. En revanche, elle requiert, d'une part, la modification du code d'applications potentiellement très complexes (par exemple applications de calcul intensif telles que des applications radar), d'autre part, d'effectuer cette action pour chaque application. Une autre méthode possible consiste à créer une pile *socket* alternative automatisant l'encapsulation lors de l'envoi de données. Nous avons utilisé une troisième méthode. Dans notre cas, nous avons sélectionné des applications utilisant le même intergiciel de communication, Data Distribution Service (DDS) [Object Management Group, 2014], puis avons modifié son code source pour y implémenter cette encapsulation. Lorsqu'une application envoie une donnée via DDS, cette donnée est automatiquement interceptée et transmise à Ptolemy II.

Hybridation inférieure : extensions Ptolemy II

La couche d'hybridation inférieure est constituée des extensions Ptolemy II suivantes :

— `EricaAppPart` : écoute les paquets entrants, les étiquette avec un identifiant d'application, puis les transmet à `EricaMapper`. Effectue les opérations en sens inverse lors de la remontée ;

— `Erica Mapper` : associe l'identifiant d'application au nœud correspondant et vice versa ;

— `EricaHostPart` : rejoue les paquets sortants depuis la section « processeur » du modèle de nœud et transmet les paquets reçus à `Erica Mapper` pour remontée vers la couche applicative.

Le principe général de cette couche est de centraliser l'opération de mappage sur la base d'une matrice de mappage, indiquant sur quel nœud est située quelle application. Des tunnels virtuels entre cet acteur central et les nœuds sont ensuite empruntés par le trafic. Toutes les applications sont liées à tous les nœud par le biais d'acteurs de type *publish-subscribe* (Figure 5.8). Un tel acteur est associé à un canal. Les acteurs de type *subscriber* reçoivent les données publiées par les acteurs *publisher* liés au même canal. Ce principe nous permet de lier la couche d'hybridation

à tous les nœuds simulés, sans besoin de les relier explicitement dans le modèle ni de modifier la couche d'hybridation en fonction de la topologie du réseau.

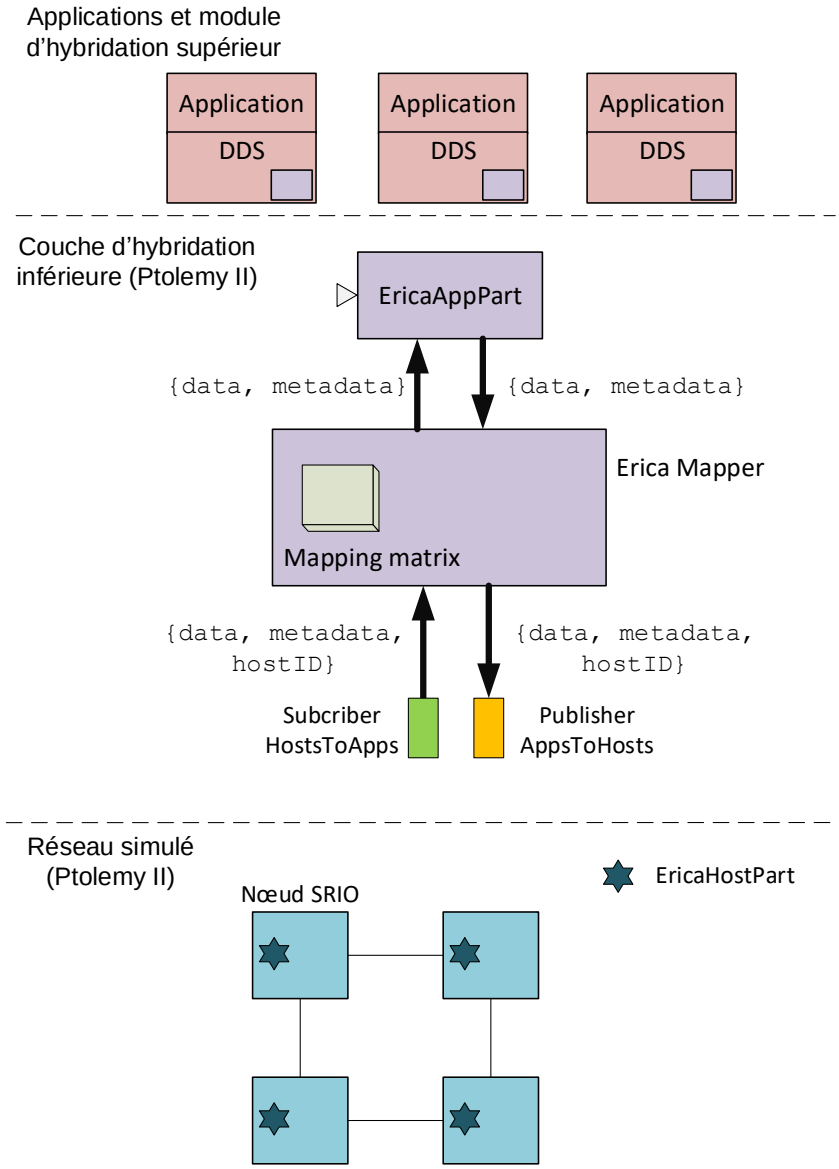

FIGURE 5.8 – Couche d'hybridation inférieure dans Ptolemy II

L'acteur `EricaAppPart` écoute les paquets en provenance des applications, extrait leurs méta-données (adresses IP, ports, et identifiant unique d'application) puis crée l'entité Ptolemy II correspondant au paquet, associée à ses méta-données. Il transmet ensuite ce paquet à l'acteur de mappage `Erica Mapper`. A la remontée, il reconstruit le paquet IP original grâce aux méta-données et le transmet à l'application réelle destinataire. Ceci rend l'hybridation complètement transparente du point de vue des applications.

L'acteur `Erica Mapper` est composé d'une matrice et d'un script de mappage. A la descente, il étiquette le paquet avec l'identifiant de nœud contenu dans la matrice de mappage, puis le transmet à tous les nœuds via le canal de *publish-subscribe*. Il est ensuite filtré par l'acteur `EricaHostPart` (Figure 5.9), qui ne le laisse passer que si l'étiquette correspond à son identifiant,

et le détruit sinon. A la remontée, `EricaHostPart` transmet le paquet à `Erica Mapper` par un autre canal *publish-subscribe*.

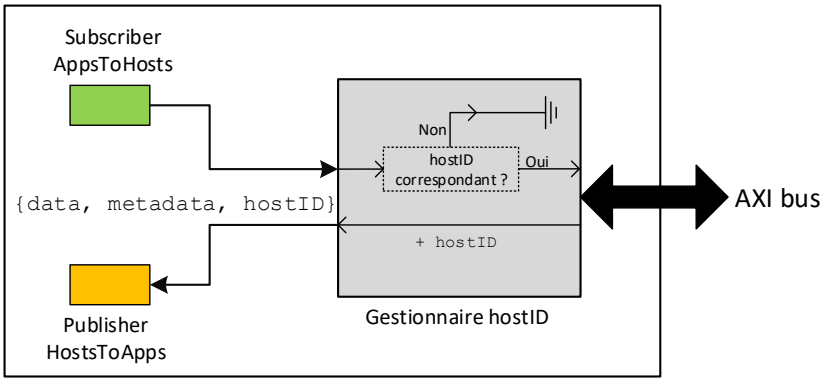

FIGURE 5.9 – Acteur EricaHostPart

La Figure 5.10 illustre un exemple simple. L'application A est située sur le nœud 1 et envoie un message à l'application B située sur le nœud 2. On constate qu'il suffit de modifier la matrice de mappage pour inverser les positions des applications et le sens de communication.

Cette architecture découple les couches applicatives et réseau et centralise la configuration du mappage applications/nœuds. Cela permet une reconfiguration rapide, facilitée, et donc une efficacité dans l'exécution de tests itératifs.

5.4.5 Synchronisation temporelle

L'hybridation nous permet de faire communiquer des applications réelles sur le réseau simulé, mais ce couplage n'est pas suffisant pour obtenir un comportement réaliste du système.

Si l'on prend l'exemple de la latence, le simulateur est capable de calculer des valeurs précises ; par exemple, que l'envoi du paquet de données de l'application A à l'application B prendra un temps t. Cependant, la simulation de cette communication peut être plus longue que la latence calculée, surtout dans le cas des systèmes embarqués où l'on étudie des latences très faibles. Cela signifie que le temps entre le moment où l'application A envoie le paquet et le moment où l'application B le reçoit n'est pas nécessairement celui qui est calculé par la simulation. Il dépend en fait du temps de mise en œuvre de l'hybridation et de la simulation de la communication.

Or, pour que notre méthode soit pertinente, il est essentiel que la latence ressentie par les applications soit la même que celle calculée par la simulation. Plus généralement, le comportement réseau ressenti par les applications doit être cohérent avec la simulation. Ce problème de synchronisation temporelle est un problème classique dans le domaine de la co-simulation. [Fujimoto, 2001] et [Perumalla, 2006] synthétisent les défis et techniques usuelles relatifs à cette problématique. Il s'agit d'un problème complexe, que nous n'avons pas souhaité étudier en profondeur, étant donné qu'il ne s'agit pas de l'objectif de cette thèse. Nous souhaitions une solution simple à implémenter et rapidement fonctionnelle pour mettre en œuvre l'architecture ERICA. Il est envisageable, à l'avenir, d'améliorer la méthode de synchronisation temporelle grâce à la littérature et des recherches supplémentaires.

Nous proposons une approche originale mais simple à faire fonctionner, directement applicable dans ERICA et conservant la transparence de l'hybridation du point de vue des applications. Le principe est que les applications avancent au rythme de la simulation, mais sans se synchroniser activement avec elle. Pour ce faire, on va faire en sorte que l'exécution des applications soit

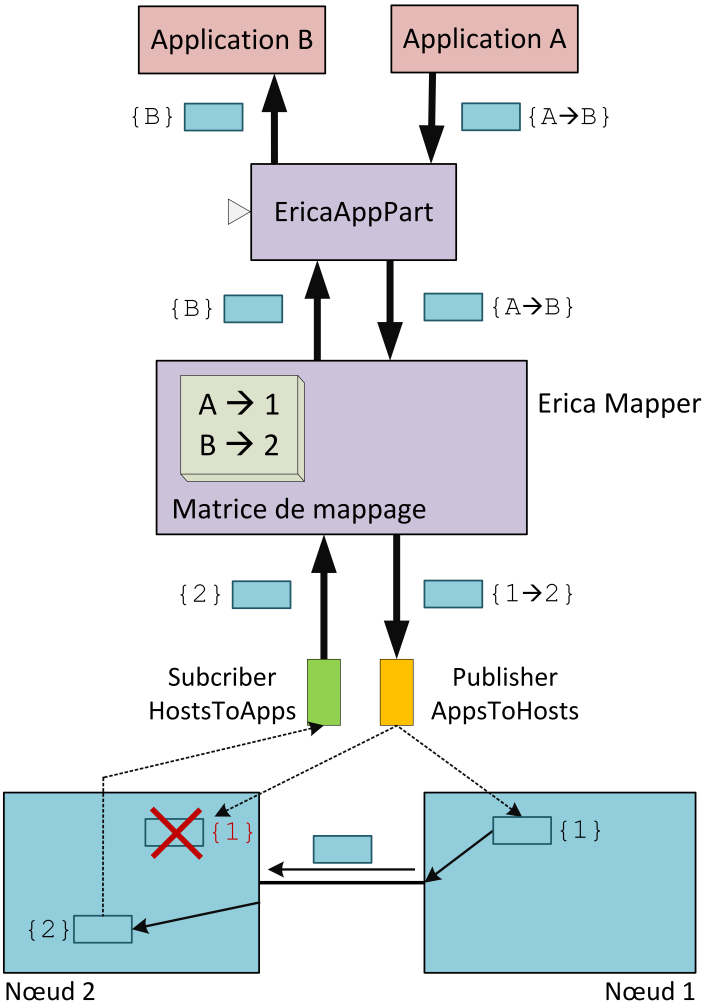

FIGURE 5.10 – Exemple du mécanisme d'hybridation

rythmée par une horloge directement contrôlée par la simulation, plutôt que l'horloge physique du système hôte. Afin de remplacer l'horloge rythmant les applications, on embarque ces dernières dans des machines virtuelles, dont l'horloge virtuelle sera une horloge contrôlée par la simulation.

Par défaut, l'horloge virtuelle trotte à une fréquence proportionnelle à la fréquence de l'horloge système : à la même fréquence, plus lentement, ou plus rapidement. Lorsque le temps de la simulation est trop éloigné du temps de l'horloge virtuel (l'écart seuil étant un paramètre), la simulation met en pause l'horloge virtuelle. Lorsqu'elle la rejoint, elle la remet en marche. Enfin, lorsqu'elle la dépasse d'un écart trop important, elle l'attend ; la simulation se met donc en pause. Ce comportement est illustré en Figure 5.11. Dans notre cas, il est embarqué dans un gestionnaire d'horloge sous la forme d'un acteur Ptolemy II, *Virtual Clock Manager*.

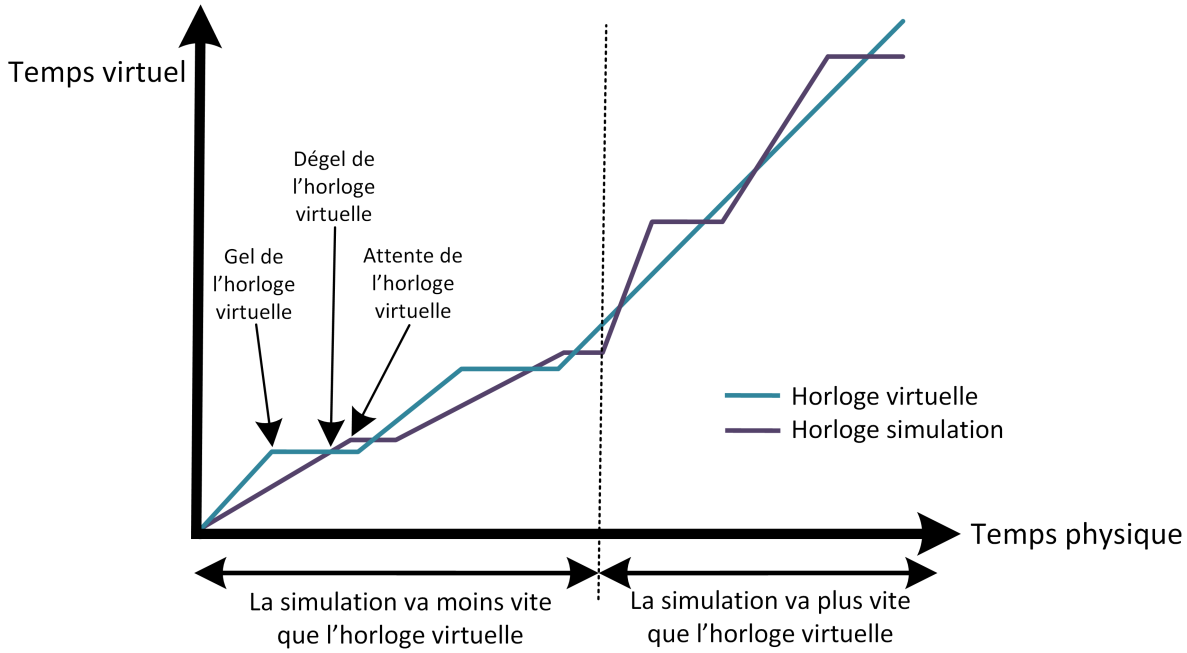

FIGURE 5.11 – Exemple de progression des différentes horloges

Les applications sont ensuite directement esclaves de cette horloge virtuelle. Ainsi, elles vont naturellement avancer à un rythme proche de la simulation, modulo quelques écarts explicités précédemment et paramétrés par l'utilisateur. Si ce dernier exige une grande précision de l'expérimentation, il fera, dès le début, trotter l'horloge virtuelle plus lentement que l'horloge système. Cela minimisera l'impact relatif des latences de réaction du *Virtual Clock Manager*. De plus, l'écart seuil déclenchant les différentes mises en pause (gel) sera réduit. Ainsi, ces paramètres seront ajustés en fonction de la précision exigée. Ceci accorde une bonne flexibilité à cette méthode et permet à l'expérimentation d'avancer le plus vite possible, contrairement, par exemple, à une approche où tous les acteurs avanceraient de façon synchrone sur de petits intervalles.

Cette approche de synchronisation temporelle est résumée en Figure 5.12. Nous avons réalisé une preuve de concept de cette approche via l'hyperviseur QEMU [QEMU, 2017]. Dans cet hyperviseur, le temps courant est obtenu via une fonction *getRealTime*. Nous avons modifié cette fonction pour qu'elle retourne le temps courant de l'horloge virtuelle à la place. L'acteur Ptolemy II *Virtual Clock Manager* a également été implémenté et l'horloge virtuelle est un programme simple écrit en C. [Quaglia, 2011] propose d'autres approches de synchronisation entre QEMU et un simulateur. Elles supposent cependant une participation active de l'hyperviseur

aux actions de synchronisation. Dans notre cas, la synchronisation est transparente du point de vue de l'hyperviseur.

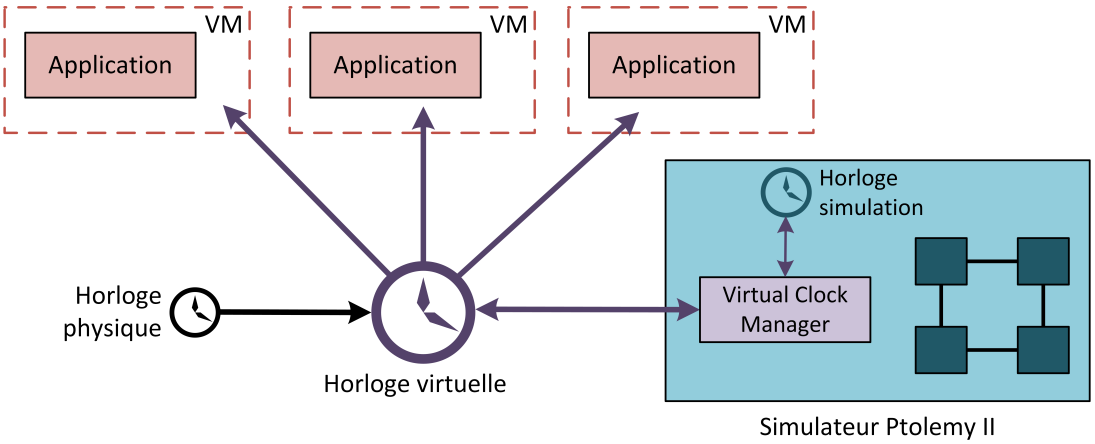

FIGURE 5.12 – Illustration du mécanisme de synchronisation temporelle

Notons que l'utilisation de machines virtuelles comme hôtes des applications permet également d'émuler des processeurs correspondant aux processeurs réellement utilisés, ce qui tout aussi important à la pertinence de l'expérimentation.

5.4.6 Mise en œuvre de la symbiose

Après avoir vérifié que le simulateur fonctionnait correctement d'un point de vue protocolaire, l'utilisation du principe de symbiose nous permet de régler les latences calculées et vérifier qu'elles sont cohérentes avec les valeurs mesurées dans la réalité. Nous utilisons des applications simulées dans Ptolemy II afin de nous baser sur des communications très simples et maîtrisées.

Certaines latences sont constantes, quand d'autres augmentent de façon linéaire avec la taille du paquet. Pour chaque section clé, on mesure donc la latence correspondante pour des données de différentes tailles. Par interpolation, on en déduit une fonction affine calculant la latence en fonction de la taille. Un exemple est donné en Tableau 5.1 pour la section clé « commutateur vers tampon d'émission ».

Une fois cette étape effectuée pour chaque section clé, on recommence le travail de comparaison des latences bout-en-bout, pour des envois simples de données de tailles différentes (Tableau 5.2). Ces mesures valident notre méthode de construction d'une simulation pertinente par la composition de sections clés et la comparaison itérative avec une plateforme matérielle.

Le processus symbiotique peut être poursuivi avec des scénarios de complexité croissante, jusqu'à obtenir satisfaction avec la simulation. Dans notre cas, nous avons notamment comparé les valeurs de latences sur des scénarios mettant en œuvre l'attente des paquets au sein des files d'attente du commutateur. Il est important de comprendre qu'il s'agit d'un processus flexible et évolutif. Le type et le nombre de scénarios utilisés dans le processus symbiotique dépendent en effet de l'exigence de précision et de la complexité de l'architecture.

5.4.7 Automatisation

Une première version du *framework* ERICA a été réalisée via un stage encadré à Thales Research & Technology. Elle a été effectuée avec l'outil logiciel Eclipse Sirius [Eclipse, 2017]. Ce

TABLE 5.1 – Latences mesurées et fonction affine déduite pour la section clé commutateur vers tampon d'émission

Taille de donnée	**Latence (ns)**
32 bit	296 ns
256 bit	232 ns
1200 bit	711 ns
(paquet rempli) 1984 bit	1096 ns
Fonction déduite	$\mathbf{L(S) = 0,5 \times S + 107}$ où L(S) est la latence pour une donnée de S bits en nanosecondes

TABLE 5.2 – Latences bout-en-bout pour l'envoi simple de données de tailles différentes

Taille de donnée	**Ptolemy II**	**Plateforme**
256 bit	2222 ns	2099 ns
800 bit	4221 ns	4113 ns
1200 bit	5512 ns	5436 ns
1984 bit	8248 ns	8226 ns
$\overline{\mathbf{\Delta}}$	**2,5%**	

dernier permet de créer simplement un atelier de modélisation graphique sur-mesure : palette graphique, modèle de données lié, fenêtres de configuration, etc.

Cette première version d'ERICA permet de gérer des bassins d'applications, créer des réseaux RapidIO, configurer les nœuds du réseau (tables de routage statiques par exemple), puis déposer les applications sur les nœuds par glisser-déposer. Enfin, deux méthodes de déploiement ont été implémentées, l'une sur simulateur et l'une sur plateforme. La première génère le fichier XML de simulation Ptolemy II correspondant à la topologie dessinée, puis configure la couche d'hybridation en cohérence. La deuxième déploie les applications sur la plateforme matérielle. Dans les deux cas, une fenêtre de lancement des applications est ensuite générée, qui permet de lancer l'exécution d'une ou plusieurs applications en même temps. Enfin, les modèles et bassins d'applications peuvent être sauvegardés et chargés dans l'environnement ERICA.

La Figure 5.13 montre une capture d'écran de la première version du logiciel, et la Figure 5.14 le résultat de la génération.

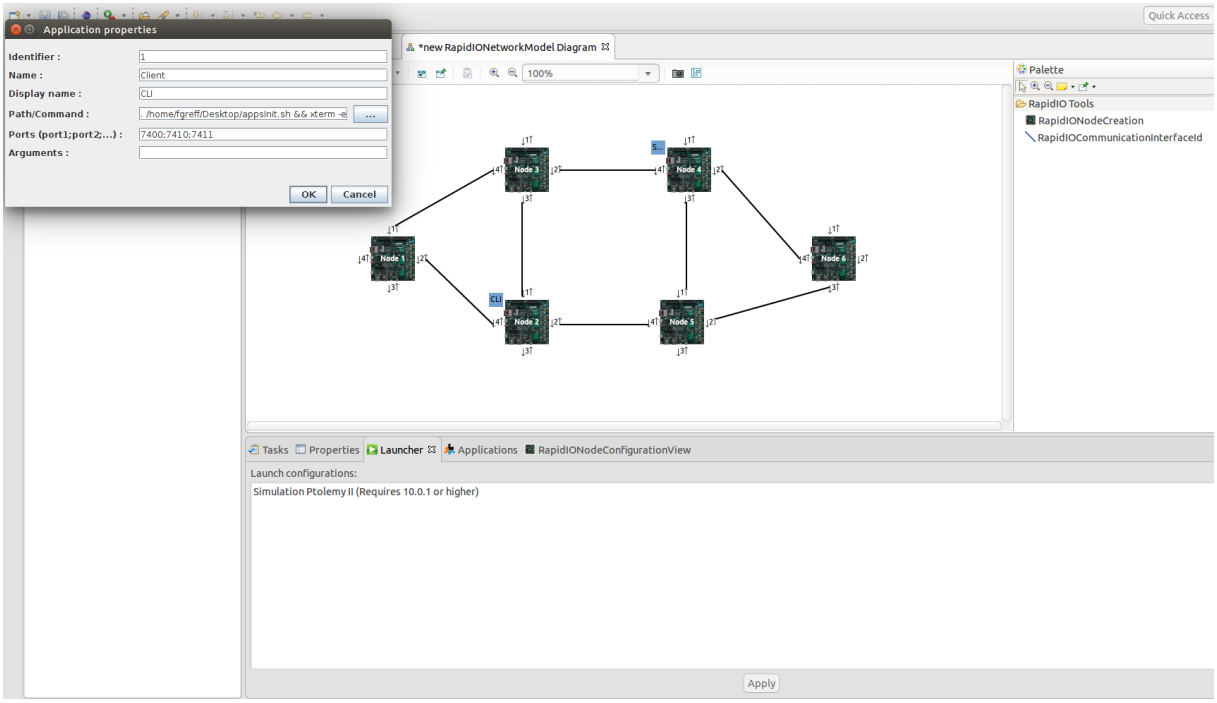

FIGURE 5.13 – Capture d'écran de la première version du logiciel ERICA

5.4.8 Bilan

La méthode ERICA nous a permis de réaliser un simulateur pertinent de notre plateforme RapidIO, qui a été utilisé pour la conception du protocole SDRN.

La confrontation systématique du modèle de simulation théorique et de l'implémentation matérielle permet d'assurer la pertinence de la simulation, d'augmenter son niveau d'abstraction et de déceler des erreurs. Dans notre cas, cette approche a notamment permis de déceler une erreur importante dans l'implémentation matérielle. Cette erreur augmentait la latence d'un niveau relatif non négligeable, mais d'un niveau absolu difficile à percevoir car faible (microsecondes). La confrontation systématique de l'implémentation matérielle et du modèle de simulation a permis de constater puis comprendre cette différence.

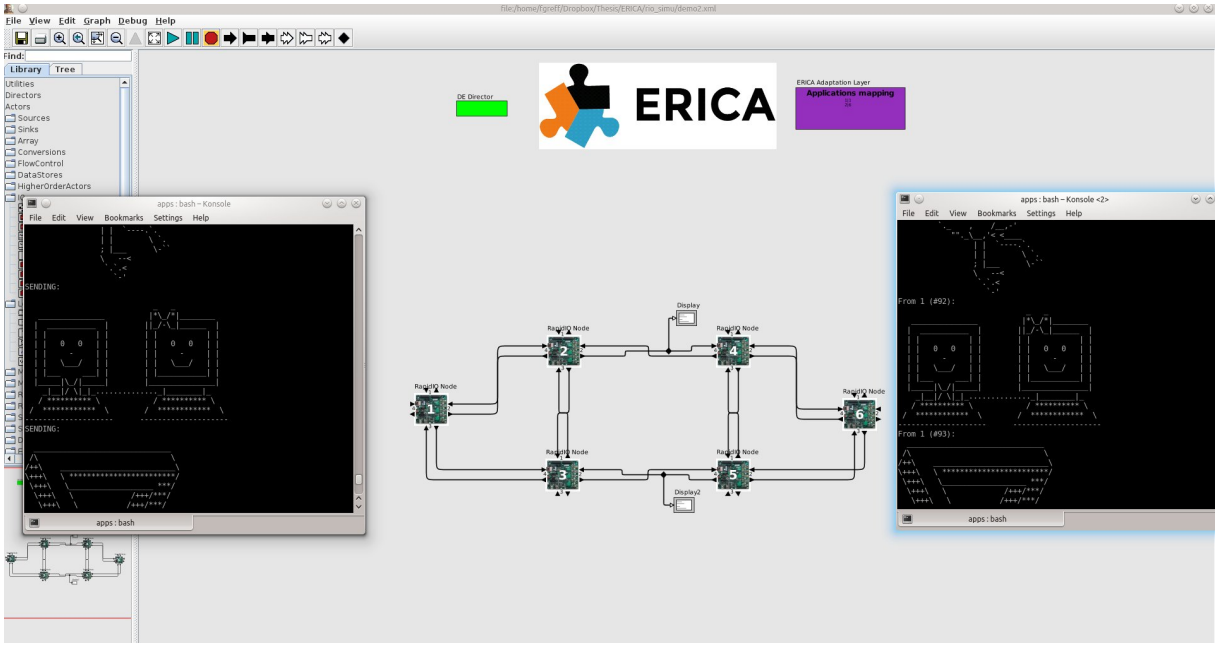

FIGURE 5.14 – Capture d'écran de la simulation générée par ERICA

La simulation est alors conçue par composition de sections significatives dont le fonctionnement exact n'est pas simulé, mais dont le résultat de simulation est similaire à la réalité. Enfin, l'hybridation entre les couches logicielles réelles et la couche simulée renforce ce lien entre les deux mondes et permet de réduire l'expérimentation réseau à son utilité principale, à savoir la conception et/ou l'observation de mécanismes réseau.

L'utilisation d'un outil graphique pour la mise en place des piles d'expérimentation nous a permis de constater la valeur ajoutée de l'automatisation des expérimentations. L'automatisation réduit en effet drastiquement le temps de mise en place des expérimentations et améliore la portabilité des acteurs de simulation (puisque tous les fichiers nécessaires sont générés). Elle fluidifie également la pensée du processus d'expérimentation, celle-ci s'effectuant sur une vue haut niveau et fonctionnelle des architectures d'expérimentation, dont la complexité d'implémentation est masquée de l'utilisateur.

5.5 Générisation de l'approche : le framework ERICA

Après avoir mis en œuvre cette première version d'ERICA, nous avons générisé l'architecture logicielle afin qu'elle corresponde à l'architecture illustrée en Figure 5.2.

Le modèle logiciel est donné en Figure 5.15. Nous en détaillons maintenant les différents composants.

5.5.1 Applications

Les applications possèdent quelques paramètres : un identifiant, un nom, un chemin vers l'exécutable, les numéros de port applicatifs qu'elles utilisent, ainsi que les arguments à ajouter lors de l'exécution. Ces paramètres peuvent utilisés par la couche d'hybridation, mais pas nécessairement.

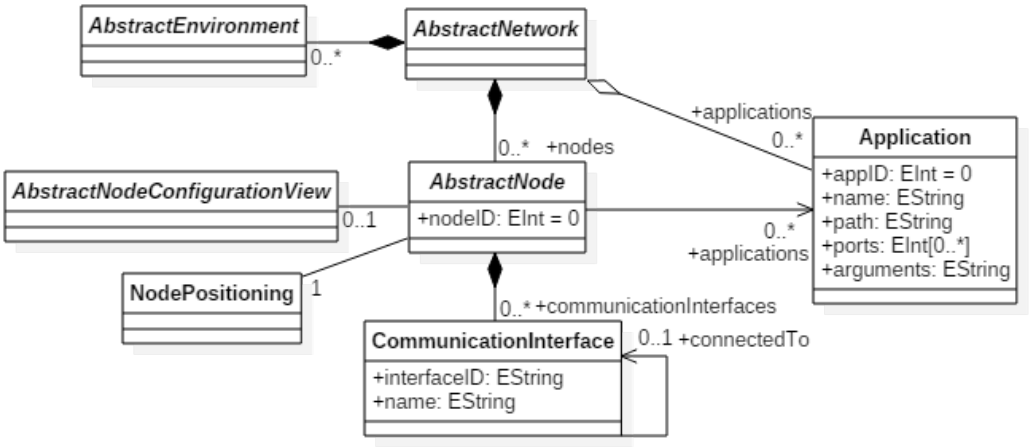

FIGURE 5.15 – Modèle UML du *framework* ERICA

Un bassin d'applications (*ApplicationPool*) est un ensemble d'applications (Figure 5.16), il peut être sauvegardé et chargé. On peut ainsi gérer différents ensembles d'applications métier, ou d'applications liés à un scénario d'expérimentation.

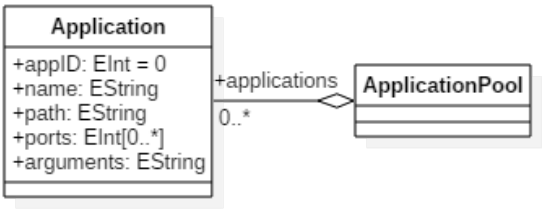

FIGURE 5.16 – Modèle UML des bassins d'application

5.5.2 Réseau et nœuds

Le réseau est un ensemble de nœuds reliés par des interfaces de communication. Le réseau contient des applications, placées ou non sur des nœuds(Figure 5.17).

On peut ajouter des fenêtres de configuration aux nœuds. Il s'agit d'une fonctionnalité totalement optionnelle. L'utilisateur implémentant un nouveau type de réseau est libre d'associer une ou plusieurs fenêtres de configuration aux nœuds de son réseau. L'environnement de modélisation d'Eclipse Sirius permet une intégration graphique automatique de ces fenêtres au sein de l'environnement ERICA.

On peut également définir une méthode d'organisation graphique des nœuds et de leurs interfaces de communication. Dans notre cas par exemple, la méthode positionnait automatiquement les quatre ports aux points cardinaux des nœuds.

Ces deux fonctionnalités optionnelles, les fenêtres de configuration (*AbstractNodeConfigurationView*) et les méthodes de positionnement (*NodePositioning*), sont représentées en Figure 5.18.

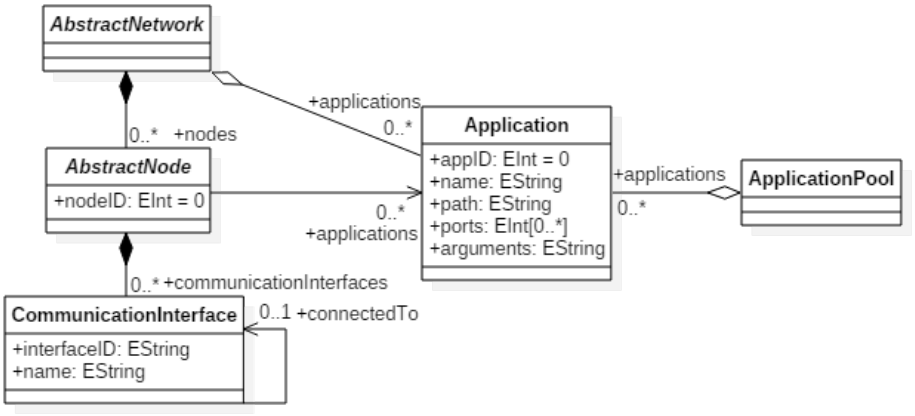

FIGURE 5.17 – Modèle UML de la partie réseau

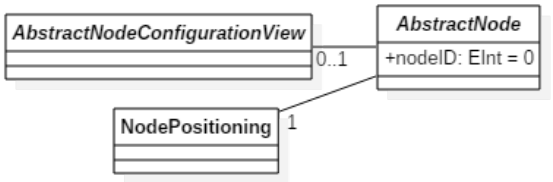

FIGURE 5.18 – Modèle UML des fonctionnalités optionnelles spécifiques aux nœuds

5.5.3 Environnement d'expérimentation

Chaque type de réseau peut être mis en œuvre au sein d'un ou plusieurs environnements d'expérimentation (Figure 5.19). Lorsqu'un utilisateur souhaite ajouter le support d'un nouvel environnement d'expérimentation pour un réseau donné, il lui suffit d'implémenter une instance d'*AbstractEnvironment*. Le reste du modèle de réseau, déjà présent, est directement utilisable.

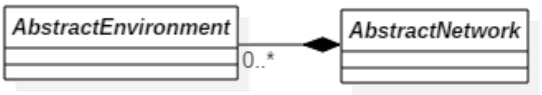

FIGURE 5.19 – Modèle UML des environnements d'expérimentation

5.5.4 Synthèse

La Figure 5.15 résume l'architecture logicielle du *framework* ERICA, mettant en œuvre les principes illustrés en Figure 5.2.

Cette architecture découple les deux parties de la couche d'hybridation. La partie haute, liée aux applications, fonctionne en effet sur tout réseau, sans avoir à la redéfinir. La gestion des bassins d'application et le mécanisme de positionnement sur les nœuds est automatiquement réutilisable sur un autre type de réseau. Il est possible d'ajouter le support d'un nouveau type de réseau en implémentant une instance d'*AbstractNetwork*. Le modèle de réseau est également découplé des environnements d'expérimentation, dont la liste ainsi que les méthodes de mise en œuvre associées (génération de code par exemple) peut évoluer sans toucher au modèle du réseau.

Par la réalisation de ce logiciel, nous avons montré la faisabilité d'une architecture générique pour l'expérimentation hybride et automatisée de réseaux.

5.6 Conclusion

La méthode d'expérimentation utilisée pour la conception de solutions réseau innovantes est en elle-même un sujet de recherche et de réflexion intéressant, mettant en lumière plusieurs défis lorsqu'on cherche à améliorer et optimiser le processus d'expérimentation. Dans ce chapitre, nous avons montré l'intérêt d'utiliser plusieurs environnements d'expérimentation au sein du même processus d'expérimentation, de découpler les environnements de l'architecture d'expérimentation, d'automatiser la mise en place des expérimentations, ainsi que d'hybrider des éléments réels et simulés dans ces dernières. Nous avons regroupé l'ensemble de ces concepts sous une méthode appelée ERICA. Elle a ensuite été pensée comme un *framework* plus générique, dédié à la mise en place d'expérimentations hybrides et multi-plateformes de réseaux, au sein d'un même environnement graphique de conception. Un point majeur de cette évolution est la possibilité pour l'utilisateur d'ajouter le support d'un nouveau réseau ou environnement d'expérimentation tout en profitant naturellement de l'ensemble des concepts d'ERICA.

Après avoir décrit les concepts de la méthode ERICA, nous avons détaillé leur mise en œuvre dans le cadre de notre étude. Des applications réelles peuvent communiquer alternativement sur une plateforme matérielle et sur une simulation Ptolemy II. L'enrichissement mutuel entre les deux environnements d'expérimentation, la symbiose, nous a permis de les améliorer et les concevoir plus rapidement. Cette mise en œuvre a posé des défis d'hybridation de la couche

applicative et la couche simulée, notamment en termes de transparence de l'hybridation essentielle à la méthode ERICA, mais également de synchronisation temporelle entre les deux couches hybridées. Une approche originale a été proposée pour répondre à ces défis. Elle utilise des acteurs *publish-subscribe* pour l'hybridation souple et transparente, et un ensemble d'horloges et machines virtuelles pour la synchronisation temporelle. Enfin, nous avons proposé une architecture logicielle de générisation de l'approche ERICA. Cette architecture a été implémentée grâce au *framework* de modélisation Eclipse Sirius. Ceci nous a permis de montrer la faisabilité de cette approche générique, mais également de disposer d'un environnement graphique complet de mise en place de nos expérimentations hybrides. Cet environnement a été utilisé pour la conception et le test du protocole SDRN.

Chapitre 6

Implémentation sur plateforme RapidIO

Sommaire

6.1 Introduction

Ce chapitre traite de l'implémentation du SDRN sur plateforme matérielle RapidIO. Ce travail permet de démontrer la faisabilité de la technologie sur un protocole pris sur étagère. Le choix du protocole RapidIO, présenté en Chapitre 1, est une contrainte industrielle de Thales Research & Technology. Ce choix est notamment lié à ses performances en termes de débit et de latence, ainsi que la présence de champs ouverts dans les entêtes protocolaires sur lesquels nous pouvons mapper des concepts du SDRN. Les nœuds du réseau sont des cartes FPGA contenant une implémentation matérielle de commutateur RapidIO, ainsi qu'un processeur sur lequel sont situées les applications. Le processeur est lié au commutateur par l'intermédiaire d'un bus AXI et d'un port logique.

La première partie du travail a consisté à implémenter le protocole RapidIO sur cette plateforme. La seconde partie a consisté à implémenter l'architecture SDRN sur la plateforme, ce qui peut être divisé en plusieurs parties :

— Interface utilisateur permettant aux applications de communiquer sur le réseau ;
— Interface de configuration et supervision du réseau depuis le contrôleur ;
— Interface de contrôle d'admission (requêtes de flux) ;
— Implémentation du contrôleur ;
— Implémentation de l'ordonnancement *Credit-Bounded Weighted Round Robin* à bas niveau.

L'architecture de l'implémentation, dont certains points seront abordés plus loin dans ce chapitre, est résumée en Figure 6.1. L'utilisation du matériel passe par un pilote SDRN. La couche SDRN doit être transparente du point de vue du matériel. Globalement, le travail d'im-

plémentation a donc demandé de concevoir les interfaces de programmation, identifier la façon dont le matériel et le protocole RapidIO peuvent être utilisés pour réaliser les différentes fonctions, puis implémenter cette réalisation dans le pilote SDRN. J'ai piloté l'ensemble de ce travail d'implémentation. Il a notamment fait intervenir deux prestataires, un apprenti et un stagiaire ingénieur.

L'objectif de ce chapitre n'est pas de donner les détails techniques de l'implémentation, mais de présenter quelques points saillants de ce travail. Nous aborderons en premier l'interface utilisateur et la technique de tunnelisation du trafic IP. Puis nous présenterons succinctement l'implémentation du contrôleur et de son interface avec le matériel. Enfin, nous présenterons l'utilisation du contrôleur SDRN au sein d'un projet collaboratif. De la documentation technique supplémentaire est donnée en annexes.

6.2 Interface utilisateur et tunnelisation du trafic IP

Cette partie concerne l'interface entre les applications et le matériel. Une fois que le flux a été alloué, elle permet aux applications d'envoyer des messages sur le réseau. Le défi principal de ce travail est la compatibilité avec des applications déjà existantes et utilisant la pile TCP/IP, dans un souci de transfert technologique. Ces applications utilisent l'interface de programmation socket et des adresses IP pour communiquer en mode connecté ou non. Pour qu'elles puissent communiquer sur le réseau SDRN, nous devons donc mapper les concepts de la pile socket sur l'architecture SDRN pour construire l'interface utilisateur. Le trafic IP sur Ethernet est ensuite tunnelisé dans le trafic SDRN sur RapidIO. A l'avenir, il pourrait être envisagé de retirer les couches TCP/IP/Ethernet de la pile et permettre aux applications de communiquer directement en SDRN via l'interface socket. Pour des raisons de simplicité sans perte de généralité, nous utilisons uniquement la version IPv4 du protocole.

Le défi est de faire le lien entre le paradigme IP, orienté machine, et le paradigme SDRN, orienté contenu. En effet, lorsqu'une application communique, elle envoie un message à une machine précise, grâce à son adresse IP. L'application destinataire est ensuite discriminée grâce au numéro de port. Or, l'architecture SDRN est orientée contenu, c'est-à-dire qu'une application ne sait pas avec quelle tuile physique elle communique. La configuration du réseau est transparente du point de vue des applications. Ces dernières peuvent également migrer régulièrement d'une tuile à une autre. Pour répondre à ce problème, nous proposons une solution divisée en deux parties :

— Les adresses IP sont utilisées pour définir des applications et non plus des machines. Le dernier octet de l'adresse correspond à l'identifiant de l'application, soit 254 applications possibles (les numéros 0 et 255 ne peuvent utilisés car ils correspondent respectivement aux identifiants de réseau et de diffusion dans le protocole IP). Lorsqu'une application de numéro X écrit à une application de numéro Y, elle écrira donc à l'adresse IP `10.0.0.Y`. Le numéro de port est inchangé et dépend de l'application. Son traitement est effectué par la couche transport de la pile TCP/IP et n'est pas modifié ;

— Pour faire le lien entre les communications IP et les flux, nous utilisons des tables de **règles SDRN**. Ces tables sont similaires aux tables de règles de pare-feu ou de commutateur OpenFlow. Le numéro de flux, ou de nœud SDRN destination dans le cas des flux non temps réel, est déduit des informations sur les numéros d'applications et de ports. Certaines informations peuvent être absentes de la table. Par exemple, le numéro de port source peut être choisi aléatoirement lors de la communication et n'avoir aucune importance dans la discrimination. Un exemple est donné en Tableau 6.1. Ce mécanisme

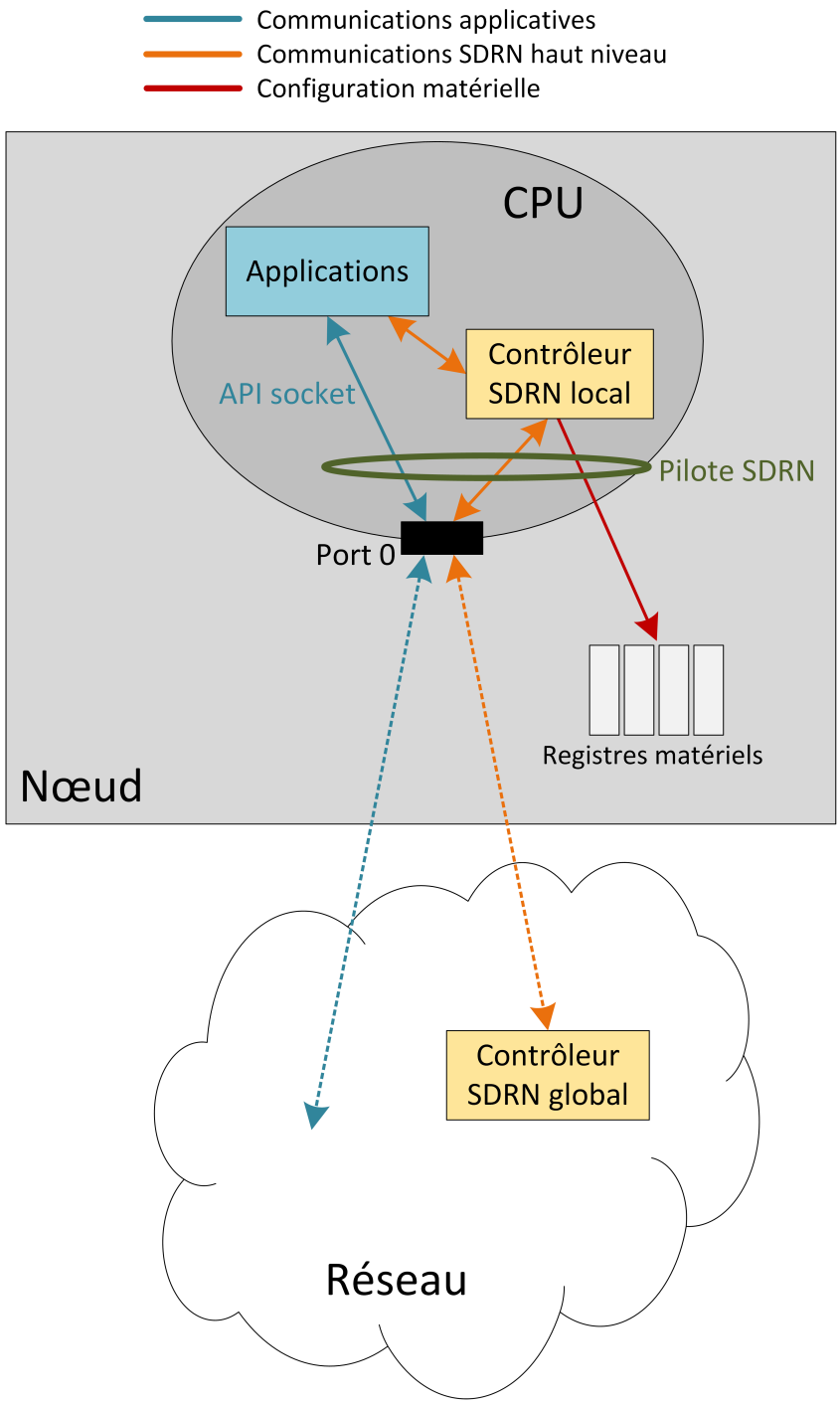

FIGURE 6.1 – Architecture de l'implémentation SDRN

TABLE 6.1 – Exemple de table de règles SDRN

Adresse IP dest.	Port source	Port dest.	Flux SDRN	Adresse SDRN dest.
10.0.2.1	*	4000	1	-
10.0.2.4	*	4001	-	2
10.0.3.1	2000	2010	-	5
10.0.3.5	*	2011	3	-

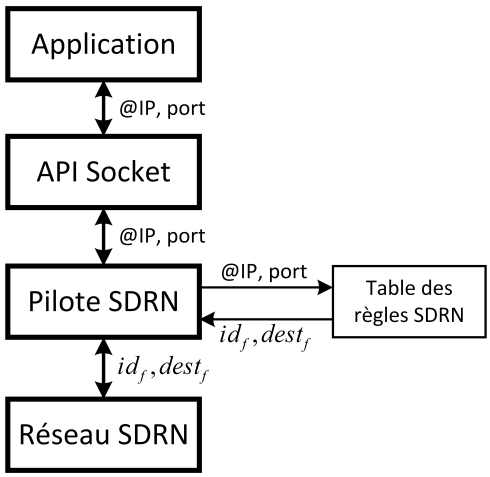

FIGURE 6.2 – Architecture de l'interface de communication

nécessite l'information sur l'application source. Nous proposons de l'embarquer également dans l'adresse IP. Ainsi, toujours dans un scénario de communication de X vers Y, l'adresse IP utilisée devient `10.0.X.Y`. A partir de cette adresse IP et des numéros de ports, la couche de mappage présente dans le pilote SDRN est capable de déduire le flux SDRN correspondant à la communication. Ce mécanisme permet également à l'application destinataire de retrouver facilement l'adresse IP qu'elle doit utiliser pour répondre ; il lui suffit en effet d'inverser les deux derniers octets.

Une illustration de l'interface utilisateur est donnée en Figure 6.1. Une documentation technique plus détaillée sur l'interface de programmation est donnée en Annexe A.

Un dernier point à aborder est l'utilisation des priorités en mode non temps réel (*best-effort*). En effet, nous avons envisagé dans l'architecture SDRN que les applications communiquant en mode non temps réel puissent préciser un niveau de priorité dans un objectif de Qualité de Service. Le protocole RapidIO propose 4 bits d'entête pouvant être utilisés pour spécifier un niveau de priorité. Puisque le niveau de priorité maximal est réservé au mode recouvrement (cf. Chapitre 3), cela laisse 15 niveaux de priorité aux flux non temps réel. Malheureusement, l'interface socket ne permet pas de spécifier de niveau de priorité. Il existe bien un drapeau « urgent » (URG), mais celui-ci est ignoré par les couches basses et n'est supporté que par d'anciennes implémentations. Un moyen d'utiliser les niveaux de priorité dans le réseau SDRN est donc de pré-configurer les flux non temps réel désirant utiliser cette fonctionnalité. Ceci fait perdre en souplesse mais reste envisageable. Une autre solution est d'utiliser un octet de l'adresse IP pour spécifier le niveau de priorité. Cette information est alors interceptée par la couche de mappage qui injecte alors le

niveau de priorité désiré dans l'entête RapidIO.

6.3 Contrôleur et configuration

L'implémentation du contrôleur, élément central du réseau SDRN, peut être divisée en trois parties majeures :

— Le cœur du contrôleur implémente les fonctionnalités de contrôle d'admission et gestion du réseau ;
— L'interface avec les applications lui permet d'échanger sur les demandes d'allocations de flux (interface *northbound*) ;
— L'interface avec le matériel lui permet de configurer et superviser le réseau (interface *southbound*).

Les deux premiers items ont été implémentés au fur et à mesure de la conception de l'architecture SDRN et ont servi à son évaluation. Cette implémentation a été présentée dans les Chapitres 2 et 3. Elle inclut les mécanismes de gestion des erreurs de transmission décrits en Chapitre 4. L'Annexe B présente l'architecture logicielle du contrôleur, et récapitule la façon dont il doit être utilisé depuis une couche de gestion des applications.

Le troisième item est lié à la plateforme matérielle. Le pilote SDRN doit permettre au contrôleur de configurer le réseau mais également d'obtenir des informations sur son évolution (par exemple, panne d'un nœud). En réalité, le contrôleur ne configure pas directement les nœuds à distance. En effet, chaque nœud possède une instance au repos de contrôleur dit contrôleur local, fonctionnant par défaut en mode esclave. L'un d'entre eux est désigné contrôleur global ; c'est cette instance qui regroupe les informations sur le réseau et le gère. Seul le contrôleur local est capable de configurer son propre nœud, pour des raisons de sécurité. Ainsi, le contrôleur global donne des ordres de configuration haut niveau aux contrôleur locaux. Ces derniers configurent alors le matériel via le pilote SDRN et l'interface de configuration. Par exemple, si le contrôleur global décide d'allouer un flux de communication sur un chemin de trois nœud, il ordonne à chacun des trois contrôleurs locaux concernés d'ajouter l'entrée correspondante dans leur table de routage. Il en est de même, en sens inverse, pour la supervision. Ce mécanisme est illustré sur la Figure 6.1.

Un canal virtuel est réservé pour les communications entre instances de contrôleurs. Sans garantir de temps réel précis, il permet néanmoins de s'assurer que les communications liées à la configuration dynamique du réseau, nécessaires à son bon fonctionnement, ne subiront pas de famine. Une autre possibilité est de réserver le deuxième niveau de priorité non temps réel pour ces communications (pour rappel, le premier niveau est réservé pour le mode recouvrement). Dans les deux cas, elles profitent de la tolérance aux pannes (routage dynamique et retransmissions) du réseau.

L'interface de configuration et de supervision permet à un contrôleur local de configurer ses tables de routage, tables de paramètres des flux pour l'ordonnancement, ainsi qu'obtenir des informations de panne. Son détail est donné en Annexe A.

6.4 Utilisation du contrôleur au sein du projet collaboratif S3P

Ceci n'est pas directement lié à l'implémentation de SDRN sur plateforme RapidIO mais concerne son utilisation en contexte industriel. Pour cette raison, nous y consacrons quelques mots dans ce chapitre.

Le contrôleur SDRN est utilisé au sein d'un projet collaboratif français en cours, S3P pour *Smart, Safe and Secure Platform*. Cette plateforme vise à permettre le développement et l'exploitation commerciale rapide de services et de produits connextés à l'Internet des Objets, combinant des caractéristiques uniques de sûreté de fonctionnement, de cybersécurité, d'agilité et de portabilité. Au sein de ce projet, Thales participe notamment à un *work package* concernant un cas d'utilisation vétronique. L'objectif est de montrer la faisabilité d'une reconfiguration d'un réseau maillé embarqué dans un véhicule, en se concentrant sur les aspects modélisation et outillage logiciel. Le réseau étudié supporte notamment des flux vidéo de caméras et des applications de traitement d'images. Le *work package* se concentre sur les aspects modélisation du système et de ses reconfigurations, mais utilise le contrôleur SDRN pour vérifier la faisabilité des configurations et le routage associé.

Une documentation des fonctions et interfaces créées spécifiquement pour ce projet est donnée en Annexe B.

6.5 Conclusion

Ce travail d'implémentation nous a permis de montrer la faisabilité de SDRN dans un contexte réel. Il a également mis en lumière quelques problématiques comme la compatibilité entre un réseau SDRN et des applications basées sur le paradigme IP. Pour y répondre, nous proposons d'utiliser les adresses IP comme identifiants de couples d'applications, puis une table de règles pour déduire les flux SDRN liés aux communications.

Le résultat de ce travail est satisfaisant d'un point de vue scientifique puisque nous avons réussi à implémenter et faire fonctionner l'ensemble de l'architecture SDRN sur une plateforme matérielle. En revanche, cette dernière souffre encore à l'heure actuelle de problèmes d'optimisation importants, réduisant considérablement sa performance. Il ne nous a ainsi pas été possible de profiter des hautes performances (plusieurs dizaines de Gbit/s) que nous envisagions avec l'utilisation du protocole RapidIO. Ces problèmes sont liés à la fois à l'optimisation de l'implémentation, à l'utilisation d'une partie de la pile TCP/IP à haut niveau, mais également à la puissance des cartes et processeurs que nous avons utilisés. Malgré cette constatation, ce travail boucle la recherche initiale sur SDRN et nous permet de proposer une solution bout-en-bout, incluant la réponse à certaines considérations industrielles.

Bilan de thèse et perspectives

Ce document a restitué les travaux de thèse concernant une nouvelle architecture embarquée, composée d'un maillage des composants et reconfigurable dynamiquement. Cette architecture permet une meilleure optimisation de l'utilisation du réseau grâce à la reconfiguration, et une meilleure tolérance aux fautes grâce au maillage et la mutualisation de la redondance. Cette montée en performances est adaptée à l'augmentation des besoins en ressources de calcul et communication des nouvelles applications embarquées, traitant des données de plus en plus volumineuses. Dans ce contexte, nous avons étudié un réseau embarqué qui permet la communication entre tous les éléments d'une telle architecture. Ce réseau, au service de la couche applicative, doit s'adapter au mieux aux demandes effectuées à l'exécution, tout en garantissant le respect des contraintes temps réel aux applications critiques. Le défi général de nos travaux a consisté à coupler les mécanismes de routage et configuration dynamique que l'on peut retrouver dans les réseaux non critiques, avec les mécanismes de sûreté et de temps réel propres au domaine de l'embarqué.

La contribution majeure de la thèse est l'architecture *Software-Defined Real-time Networking*, ou SDRN. Il s'agit d'une architecture de réseau défini par logiciel qui sait coupler les notions de temps réel, configuration et routage dynamique. Le manuscrit décrit l'architecture bout-en-bout, les algorithmes d'allocation des flux de communication, le routage dynamique, les mécanismes de tolérance aux fautes et les réponses à certaines problématiques propres à l'implémentation. Il s'agit de la première solution, à notre connaissance, à traiter l'ensemble de ces problématiques de bout-en-bout. Le premier avantage majeur de SDRN est sa modularité. En effet, nous avons proposé une solution concrète complète de mise en place d'un tel réseau, mais certains éléments, tels que la recherche de chemins, peuvent être remplacés par d'autres algorithmes. Il est donc possible de faire évoluer et améliorer l'architecture sans remettre en cause le reste de l'existant. L'aspect modulaire le plus intéressant est sans doute la possibilité de faire fonctionner le SDRN sur n'importe quel protocole bas niveau, à supposer qu'il respecte quelques propriétés que nous avons définies. Nous pouvons également citer la prise en compte de la fragmentation. Nous pensons que cette généralisation de l'approche facilitera son transfert technologique. Le deuxième avantage est l'allocation incrémentale des flux de communication. Cette propriété linéarise la complexité d'allocation et améliore son passage à l'échelle. Le troisième avantage est lié au module concret d'ordonnancement que nous proposons d'utiliser, *Credit-Bounded Weighted Round Robin* (CBWRR). Cet ordonnancement évite les pertes de bande passante dues à la non utilisation du canal virtuel par les flux temps réel. Il permet une utilisation optimale du réseau et une réduction du pessimisme lié à l'allocation incrémentale.

Dans l'architecture SDRN, il est possible de remplacer le module d'ordonnancement par un autre, du moment qu'il respecte trois propriétés : isolation, déterminisme et équité minimale. Ces propriétés permettent l'allocation incrémentale. Nous nous sommes rendu compte dans nos recherches qu'il était difficile de concevoir un tel ordonnancement. Les approches basées sur des

crédits ou leur équivalents temporels (*dynamic TDMA*) semblent les plus simples. Il devient plus difficile, par exemple, d'obtenir un ordonnancement respectant les trois propriétés à partir d'un système de priorités. Nous avons d'ailleurs observé dans la littérature que les ordonnancements temps réel basés sur des priorités nécessitaient de recalculer une partie de l'ordonnancement à chaque allocation. Ces considérations nous amènent à penser qu'il serait difficile d'adapter le SDRN à un réseau Ethernet basé uniquement sur les priorités (ou numéros de VLAN), sans mécanisme supplémentaire comme le CBWRR.

En termes d'implémentation, nous avons constaté que l'implémentation matérielle du CBWRR n'était pas facile. Elle correspond en effet à avoir une queue par sous-canal virtuel dans le pire cas. Ceci demande donc une utilisation judicieuse des tampons matériels disponibles. Nous avons également constaté que la compatibilité avec des applications IP existantes nécessitait une traduction entre le paradigme orienté machines de la pile TCP/IP et le paradigme orienté contenus du SDRN. Malgré ces points de difficulté, nous avons montré la faisabilité du SDRN sur plateforme RapidIO réelle.

D'un point de vue industriel, il est peu raisonnable de considérer un remplacement complet d'architectures embarquées par une architecture SDRN. En revanche, nous envisageons à moyen terme que le SDRN puisse être utilisé pour remplacer des sous-systèmes dédiés aux calculs hautes performances, par exemple une plateforme radar ou de traitement d'images. La modularité du protocole bas niveau pourrait faciliter une telle transition. Dans le domaine de la recherche et technologie, SDRN sera présenté courant 2018 à un salon Thales Research & Technology. Le contrôleur est également utilisé au sein du projet collaboratif S3P et sera présent dans les présentations de fin de projet (fin 2018) et au *S3P Alliance Day* de 2018.

Nous identifions quatre perspectives principales concernant l'architecture SDRN :

— Dans le contexte d'une architecture embarquée complètement dynamique, la couche applicative de négociation des ressources processeur utilise le SDRN. L'interface entre le contrôleur et la couche applicative a été pensée dans ce but. Cependant, une telle couche applicative fera peut-être apparaître, dans la pratique, de nouveaux besoins en termes de requêtes et dialogues avec le contrôleur. Ceci pourrait nécessiter une adaptation de la couche haute du SDRN ;
— La gestion efficace du *multicast* (communication point à multipoints). Il s'agit d'un problème complexe qui nécessite de prendre en compte les sous-chemins communs aux communications pour optimiser le routage ;
— Le retrait des couches TCP/IP du pilote SDRN, plutôt que leur tunnelisation ;
— L'étude de l'utilisation du SDRN sur un réseau Time-Sensitive Networking (TSN). La technologie TSN prend une ampleur croissante dans le domaine des réseaux embarqués. Nous pensons que le SDRN est une solution intéressante à la problématique de contrôle d'admission et configuration dynamique dans de tels réseaux.

Un deuxième volet de la thèse concerne la simulation et plus généralement les méthodes d'expérimentation sur les réseaux embarqués. Nos recherches dans ce domaine se sont cristallisées sous la forme d'un *framework* d'expérimentation hybride, ERICA. Celui-ci facilite la mise en place d'expérimentations mêlant aspects réels et simulés, à partir d'une interface graphique haut niveau. Il est également extensible en termes d'environnements d'expérimentation et types de réseaux supportés. Nous avons constaté qu'ERICA était très utile pour le test du protocole SDRN, notamment sur plateforme matérielle. Il nous a en effet permis de rapidement déployer des applications et configurations de test sur la plateforme, depuis un environnement centralisé.

Avec le recul, il s'agit d'une forme d'expérimentation définie par logicielle. Un deuxième point que nous avons constaté est l'intérêt d'ERICA en termes de démonstration. Il permet de passer facilement d'une plateforme à un simulateur, d'un scénario à un autre. Il facilite également l'adaptation de la démonstration aux questions de l'auditoire.

Nous sommes en revanche plus mitigés en ce qui concerne l'aspect évolutif du logiciel. En effet, le fait qu'ERICA se base sur Eclipse Sirius le rend plutôt lourd à faire évoluer, par exemple pour ajouter le support d'un nouvel environnement d'expérimentation. Malgré nos efforts pour rendre la complexité du logiciel la plus transparente possible, nous pensons que des recherches supplémentaires sur l'expérience utilisateur sont nécessaires pour obtenir un logiciel agréable à faire évoluer. Enfin, l'une de nos perspectives concernant nos travaux sur ERICA est de les intégrer au *framework* de multi-simulation MECSYCO [Camus et al., 2015]. Nous pensons en effet qu'ils peuvent l'enrichir avec les fonctionnalités d'hybridation réel/simulé et le support du simulateur Ptolemy II.

Bibliographie

[Abdelzaher et al., 2000] Abdelzaher, T. F., Atkins, E. M., and Shin, K. G. (2000). QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control. *IEEE Transsactions on Computers*, 49(11) :1170–1183.

[Ahrenholz et al., 2008] Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H. (2008). CORE : A real-time network emulator. In *MILCOM 2008 - 2008 IEEE Military Communications Conference*, pages 1–7.

[Ahuja et al., 1993] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. (1993). *Network Flows : Theory, Algorithms, and Applications*. Prentice-Hall, Inc.

[Alur and Dill, 1994] Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2) :183–235.

[ARINC, 1991] ARINC (1991). ARINC 651 : Design Guidance for Integrated Modular Avionics.

[ARINC, 1999] ARINC (1999). ARINC 629 : Multi-transmitter data bus, Part 1, Technical Description.

[ARINC, 2001] ARINC (2001). ARINC 429 : Digital Information Transfer System (DITS).

[ARINC, 2009] ARINC (2009). ARINC 664p7 : Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network.

[Audsley, 1991] Audsley, N. (1991). Optimal priority assignment and feasibility of static priority tasks with arbitrary start times.

[Aydt et al., 2009] Aydt, H., Turner, S., Cai, W., and Low, M. (2009). Research issues in symbiotic simulation. In *Simulation Conference (WSC), Proceedings of the 2009 Winter*, pages 1213–1222.

[Azodolmolky et al., 2013] Azodolmolky, S., Nejabati, R., Pazouki, M., Wieder, P., Yahyapour, R., and Simeonidou, D. (2013). An analytical model for software defined networking : A network calculus-based approach. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 1397–1402.

[Barbehenn, 1998] Barbehenn, M. (1998). A note on the complexity of dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers*, 47(2) :263.

[Baruah et al., 2014] Baruah, S., Chattopadhyay, B., Li, H., and Shin, I. (2014). Mixed-criticality scheduling on multiprocessors. *Real-Time Systems*, 50(1) :142–177.

[Baruah and Vestal, 2008] Baruah, S. and Vestal, S. (2008). Schedulability analysis of sporadic tasks with multiple criticality specifications. In *Euromicro Conference on Real-Time Systems'08, (ECRTS'08)*, pages 147–155.

[Bauer et al., 2010a] Bauer, H., Scharbarg, J. L., and Fraboul, C. (2010a). Improving the worst-case delay analysis of an AFDX network using an optimized trajectory approach. *IEEE Transactions on Industrial Informatics*, 6(4) :521–533.

[Bauer et al., 2010b] Bauer, H., Scharbarg, J. L., and Fraboul, C. (2010b). Worst-case end-to-end delay analysis of an avionics AFDX network. In *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, pages 1220–1224.

[Ben Slimane et al., 2015] Ben Slimane, J., René, S., Song, Y., Staples, G. S., Tsiontsiou, E., and Song, Y.-Q. (2015). Operator calculus algorithms for multi-constrained paths. *International Journal of Mathematics and Computer Science*, Vol. 10.

[Berard et al., 2010] Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2010). *Systems and Software Verification : Model-Checking Techniques and Tools*. Springer Publishing Company, Incorporated, 1st edition.

[Bernat et al., 2001] Bernat, G., Burns, A., and Liamosi, A. (2001). Weakly hard real-time systems. *IEEE Transactions on Computers*, 50(4) :308–321.

[Bishnoi, 2016] Bishnoi, R. (2016). Hybrid fault tolerant routing algorithm in NoC. In *Perspectives in Science*, volume 8 of *Recent Trends in Engineering and Material Sciences*, pages 586–588.

[Bjerregaard and Mahadevan, 2006] Bjerregaard, T. and Mahadevan, S. (2006). A survey of research and practices of network-on-chip. In *ACM Computing Surveys*, volume 38.

[Boppana and Chalasani, 1995] Boppana, R. V. and Chalasani, S. (1995). Fault-tolerant wormhole routing algorithms for mesh networks. In *IEEE Transactions on Computers*, volume 44, pages 848–864.

[Bouillard et al., 2010] Bouillard, A., Jouhet, L., and Thierry, E. (2010). Tight performance bounds in the worst-case analysis of feed-forward networks. In *2010 Proceedings of IEEE INFOCOM*, pages 1–9.

[Boyer and Fraboul, 2008] Boyer, M. and Fraboul, C. (2008). Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus. In *2008 IEEE International Workshop on Factory Communication Systems*, pages 11–20.

[Boyer et al., 2013] Boyer, M., Navet, N., Fumey, M., Migge, J., and Havet, L. (2013). Combining static priority and weighted round-robin like packet scheduling in AFDX for incremental certification and mixed-criticality support. In *5TH European Conference for Aeronautics and Space Sciences (EUCASS)*.

[Burns, 1991] Burns, A. (1991). Scheduling hard real-time systems : a review. *Software Engineering Journal*, 6(3) :116–128.

[Burns and Davis, 2014] Burns, A. and Davis, R. I. (2014). Mixed criticality systems - a review.

[Burns et al., 1999] Burns, A., Punnekkat, S., Strigini, L., and Wright, D. R. (1999). Probabilistic scheduling guarantees for fault-tolerant real-time systems. In *Dependable Computing for Critical Applications 7*, pages 361–378.

[Camus et al., 2015] Camus, B., Bourjot, C., and Chevrier, V. (2015). Combining DEVS with multi-agent concepts to design and simulate multi-models of complex systems. In *TMS/DEVS 2015*.

[Carothers et al., 1997] Carothers, C. D., Fujimoto, R. M., Weatherly, R. M., and Wilson, A. L. (1997). Design and implementation of HLA time management in the RTI version f.0. In *Proceedings of the 29th Winter Simulation Conference*, WSC '97, pages 373–380. IEEE Computer Society.

[Chang, 2000] Chang, C.-S. (2000). *Performance Guarantees in Communication Networks*. Telecommunication Networks and Computer Systems. Springer-Verlag, London.

[Charara et al., 2006] Charara, H., Scharbarg, J. L., Ermont, J., and Fraboul, C. (2006). Methods for bounding end-to-end delays on an AFDX network. In *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, pages 10 pp.–202.

[Charny and Boudec, 2000] Charny, A. and Boudec, J.-Y. L. (2000). Delay bounds in a network with aggregate scheduling. In *International Workshop on Quality of Future Internet Services*, Lecture Notes in Computer Science, pages 1–13. Springer, Berlin, Heidelberg.

[Chen et al., 2008] Chen, S., Song, M., and Sahni, S. (2008). Two techniques for fast computation of constrained shortest paths. *IEEE/ACM Transactions on Networking*, 16(1) :105–115.

[Clark et al., 1994] Clark, D., Braden, R., and Shenker, S. (1994). Integrated services in the internet architecture : an overview ; RFC 1633.

[Concer et al., 2011] Concer, N., Vesco, A., Scopigno, R., and Carloni, L. P. (2011). A dynamic and distributed TDM slot-scheduling protocol for QoS-oriented networks-on-chip. In *IEEE 29th International Conference on Computer Design (ICCD)*, pages 31–38.

[Cruz, 1991] Cruz, R. L. (1991). A calculus for network delay. II. network analysis. *IEEE Transactions on Information Theory*, 37(1) :132–141.

[Cucu-Grosjean et al., 2012] Cucu-Grosjean, L., Santinelli, L., Houston, M., Lo, C., Vardanega, T., Kosmidis, L., Abella, J., Mezzetti, E., Quiñones, E., and Cazorla, F. J. (2012). Measurement-based probabilistic timing analysis for multi-path programs. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 91–101.

[Dahmann and Morse, 1998] Dahmann, J. and Morse, K. (1998). High level architecture for simulation : an update. In *2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pages 32–40.

[Dally and Towles, 2003] Dally, W. and Towles, B. (2003). *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc.

[Davies et al., 1998] Davies, E., Carlson, M. A., Weiss, W., Black, D., Blake, S., and Wang, Z. (1998). An architecture for differentiated services, RFC 2475.

[Davis et al., 2015] Davis, R. I., Thekkilakattil, A., Gettings, O., Dobrin, R., and Punnekkat, S. (2015). Quantifying the exact sub-optimality of non-preemptive scheduling. In *2015 IEEE Real-Time Systems Symposium*, pages 96–106.

[Dawson-Haggerty et al., 2010] Dawson-Haggerty, S., Tavakoli, A., and Culler, D. (2010). Hydro : A hybrid routing protocol for low-power and lossy networks. In *First IEEE International Conference on Smart Grid Communications*, pages 268–273.

[Derasevic et al., 2013] Derasevic, S., Proenza, J., and Gessner, D. (2013). Towards dynamic fault tolerance on FTT-based distributed embedded systems. In *IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–4.

[Dijkstra, 1959] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1) :269–271.

[Dolejs and Hanzalek, 2003] Dolejs, O. and Hanzalek, Z. (2003). Simulation of ethernet for real-time applications. In *2003 IEEE International Conference on Industrial Technology*, volume 2, pages 1018–1021.

[Dorin et al., 2010] Dorin, F., Richard, P., Richard, M., and Goossens, J. (2010). Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities. *Real-Time Systems*, 46(3) :305–331.

[Duato, 1993] Duato, J. (1993). A new theory of deadlock-free adaptive routing in wormhole networks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 4, pages 1320–1331.

[Duato, 1995] Duato, J. (1995). A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 6, pages 1055–1067.

[Duato, 1996] Duato, J. (1996). A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks. In *IEEE Transactions on Parallel and Distributed Systems*, volume 7, pages 841–854.

[Eclipse, 2017] Eclipse (2017). Eclipse Sirius 5.1, open source solution for graphical modeling - http ://www.eclipse.org/sirius.

[Führer et al., 2000] Führer, T., Müller, B., Dieterle, W., Hartwich, F., Hugel, R., Walther, M., and Bosch, R. (2000). Time triggered communication on CAN. In *7th International CAN Conference*.

[Firoiu et al., 2002] Firoiu, V., Boudec, J. Y. L., Towsley, D., and Zhang, Z.-L. (2002). Theories and models for internet quality of service. *Proceedings of the IEEE*, 90(9) :1565–1591.

[Fisher et al., 2006] Fisher, N., Baker, T. P., and Baruah, S. (2006). Algorithms for determining the demand-based load of a sporadic task system. In *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, pages 135–146.

[Fujimoto, 2001] Fujimoto, R. M. (2001). Parallel and distributed simulation systems.

[Fukushima et al., 2009] Fukushima, Y., Fukushi, M., and Horiguchi, S. (2009). Fault-tolerant routing algorithm for network on chip without virtual channels. In *24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 313–321.

[Fuller, 2005] Fuller, S. (2005). *RapidIO : The Embedded System Interconnect*. John Wiley & Sons.

[Gavrilut et al., 2017] Gavrilut, V., Zarrin, B., Pop, P., and Samii, S. (2017). Fault-tolerant Topology and Routing Synthesis for IEEE Time-sensitive Networking. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, RTNS '17, pages 267–276. ACM.

[George and Hermant, 2009] George, L. and Hermant, J. F. (2009). A Norm Approach for the Partitioned EDF Scheduling of Sporadic Task Systems. In *21st Euromicro Conference on Real-Time Systems*, pages 161–169.

[George et al., 1996] George, L., Rivierre, N., and Spuri, M. (1996). Preemptive and non-preemptive real-time UniProcessor scheduling.

[Göhringer et al., 2011] Göhringer, D., Oey, O., Hübner, M., and Becker, J. (2011). Heterogeneous and runtime parameterizable star-wheels network-on-chip. In *Modeling and Simulation 2011 International Conference on Embedded Computer Systems : Architectures*, pages 380–387.

[Glass and Ni, 1996] Glass, C. J. and Ni, L. M. (1996). Fault-tolerant wormhole routing in meshes without virtual channels. In *IEEE Transactions on Parallel and Distributed Systems*, volume 7, pages 620–636.

[Greff, 2018] Greff, F. (2018). Procédé d'optimisation d'un système embarqué et dispositifs associés, brevet no. 18 00122, 8 février 2018.

[Greff et al., 2016] Greff, F., Dujardin, E., Samama, A., Song, Y.-Q., and Ciarletta, L. (2016). A Symbiotic Approach to Designing Cross-Layer QoS in Embedded Real-Time Systems. In *8th European Congress on Embedded Real Time Software and Systems (ERTS^2'16)*.

[Greff et al., 2017a] Greff, F., Song, Y.-Q., Ciarletta, L., and Samama, A. (2017a). A Dynamic Flow Allocation Method for the Design of a Software-Defined Real-Time Mesh Network. In *13th IEEE International Workshop on Factory Communication Systems (WFCS'17)*.

[Greff et al., 2017b] Greff, F., Song, Y.-Q., Ciarletta, L., and Samama, A. (2017b). Combining Source and Destination-Tag Routing to Handle Fault Tolerance in Software-Defined Real-Time Mesh Networks. In *25th International Conference on Real-Time Networks and Systems (RTNS'17)*.

[Guck and Kellerer, 2014] Guck, J. W. and Kellerer, W. (2014). Achieving end-to-end real-time quality of service with software defined networking. In *IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 70–76.

[Hassin, 1992] Hassin, R. (1992). Approximation schemes for the restricted shortest path problem. *Mathematics on Operations Research*, 17(1) :36–42.

[Heise et al., 2015] Heise, P., Geyer, F., and Obermaisser, R. (2015). Deterministic OpenFlow : Performance evaluation of SDN hardware for avionic networks. In *11th International Conference on Network and Service Management (CNSM)*, pages 372–377.

[Henia et al., 2005] Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., and Ernst, R. (2005). System level performance analysis - the SymTA/s approach. *IEE Proceedings - Computers and Digital Techniques*, 152(2) :148–166.

[Hesham et al., 2017] Hesham, S., Rettkowski, J., Goehringer, D., and El Ghany, M. A. A. (2017). Survey on real-time networks-on-chip. In *IEEE Transactions on Parallel and Distributed Systems*, volume 28, pages 1500–1517.

[Hu and Marculescu, 2004] Hu, J. and Marculescu, R. (2004). DyAD - smart routing for networks-on-chip. In *41st Design Automation Conference*, pages 260–263.

[Huang et al., 2014] Huang, H.-M., Gill, C., and Lu, C. (2014). Implementation and evaluation of mixed-criticality scheduling approaches for sporadic tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 13(4) :126 :1–126 :25.

[Humernbrum et al., 2014] Humernbrum, T., Glinka, F., and Gorlatch, S. (2014). Using Software-Defined Networking for Real-Time Internet Applications.

[IEEE, 2002] IEEE (2002). CSMA/CD access method.

[IEEE, 2008] IEEE (2008). IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems (IEEE 1588).

[IEEE, 2010] IEEE (2010). IEEE standard for modeling and simulation, high level architecture (HLA)– framework and rules. *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*, pages 1–38.

[IEEE 802.1 AVB TG, 2010] IEEE 802.1 AVB TG (2010). IEEE 802.1Qat - Stream Reservation Protocol.

[International Organization for Standardization, 1993] International Organization for Standardization (1993). Road vehicles - Controller Area Network (CAN).

[Jiang and Liu, 2008] Jiang, Y. and Liu, Y. (2008). *Stochastic Network Calculus*. Springer-Verlag, London.

[Jouy et al., 2014] Jouy, A., Yao, J., and Zhu, G. (2014). Optimal bandwidth allocation with dynamic multi-path routing for non-critical traffic in AFDX networks. In *20th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 600–607.

[Kariniemi and Nurmi, 2005] Kariniemi, K. and Nurmi, J. (2005). Fault tolerant XGFT network on chip for multi processor system on chip circuits. In *International Conference on Field Programmable Logic and Applications*, pages 203–210.

[Kemayo et al., 2013] Kemayo, G., Ridouard, F., Bauer, H., and Richard, P. (2013). Optimistic problems in the trajectory approach in FIFO context. In *IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8.

[Kim et al., 2007] Kim, J., Balfour, J., and Dally, W. (2007). Flattened butterfly topology for on-chip networks. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 172–182. IEEE Computer Society.

[Kim et al., 2005] Kim, K., Lee, S.-J., Lee, K., and Yoo, H.-J. (2005). An arbitration look-ahead scheme for reducing end-to-end latency in networks on chip. In *IEEE International Symposium on Circuits and Systems*, volume 3, pages 2357–2360.

[Kim and Kim, 2007] Kim, Y. B. and Kim, Y. B. (2007). Fault tolerant source routing for network-on-chip. In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, pages 12–20.

[Kleinrock, 1977] Kleinrock, L. (1977). Queueing Systems, Volume I : Theory. *IEEE Transactions on Communications*, 25(1) :178–179.

[Kopetz, 1997] Kopetz, H. (1997). *Real-Time Systems - Design Principles for Distributed Embedded*. Springer-Verlag.

[Kopetz and Bauer, 2003] Kopetz, H. and Bauer, G. (2003). The time-triggered architecture. *Proceedings of the IEEE*, 91(1) :112–126.

[Korkmaz and Krunz, 2001] Korkmaz, T. and Krunz, M. (2001). Multi-constrained optimal path selection. In *IEEE INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 2, pages 834–843 vol.2.

[Kreutz et al., 2015] Kreutz, D., Ramos, F. M. V., Veríssimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2015). Software-defined networking : A comprehensive survey. *Proceedings of the IEEE*, 103(1) :14–76.

[Krichen et al., 2010] Krichen, F., Hamid, B., Zalila, B., and Coulette, B. (2010). Designing dynamic reconfiguration for distributed real time embedded systems. In *10th Annual International Conference on New Technologies of Distributed Systems (NOTERE)*, pages 249–254.

[Kumar et al., 2017] Kumar, R., Hasan, M., Padhy, S., Evchenko, K., Piramanayagam, L., Mohan, S., and Bobba, R. B. (2017). End-to-end network delay guarantees for real-time systems using SDN. *arXiv :1703.01641 [cs]*.

[Kurose and Mouftah, 1988] Kurose, J. and Mouftah, H. (1988). Computer-aided modeling, analysis, and design of communication networks. *IEEE Journal on Selected Areas in Communications*, 6(1) :130–145.

[Lacage et al., 2009] Lacage, M., Ferrari, M., Hansen, M., Turletti, T., and Dabbous, W. (2009). NEPI : Using independent simulators, emulators, and testbeds for easy experimentation.

[Lasnier et al., 2013] Lasnier, G., Cardoso, J., Siron, P., Pagetti, C., and Derler, P. (2013). Distributed simulation of heterogeneous and real-time systems. In *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 55–62.

[Le Boudec and Thiran, 2001] Le Boudec, J.-Y. and Thiran, P. (2001). *Network Calculus : A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag.

[Lehoczky, 1990] Lehoczky, J. (1990). Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *11th Real-Time Systems Symposium*, pages 201–209.

[Lei Du and Herlich, 2016] Lei Du, J. and Herlich, M. (2016). Software-defined Networking for Real-time Ethernet. In *13th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 584–589.

[Leung and Merrill, 1980] Leung, J. Y. T. and Merrill, M. L. (1980). A note on preemptive scheduling of periodic, real-time tasks. *Information Processing Letters*, 11(3) :115–118.

[Li et al., 2014] Li, X., Cros, O., and George, L. (2014). The trajectory approach for AFDX FIFO networks revisited and corrected. In *IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 1–10.

[Lim et al., 2012] Lim, H. T., Herrscher, D., and Chaari, F. (2012). Performance comparison of IEEE 802.1q and IEEE 802.1 AVB in an ethernet-based in-vehicle network. In *8th International Conference on Computing Technology and Information Management (ICCM)*, volume 1, pages 1–6.

[Liu and Layland, 1973] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in hard real-time environment. *Journal of the Association for Computing Machinery*, 20 :46–61.

[Lu and Jantsch, 2007] Lu, Z. and Jantsch, A. (2007). Slot allocation using logical networks for TDM virtual-circuit configuration for network-on-chip. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 18–25.

[Malkin, 1998] Malkin, G. (1998). RIP Version 2. RFC 2453, The Internet Society.

[Martin and Minet, 2006] Martin, S. and Minet, P. (2006). Schedulability analysis of flows scheduled with FIFO : application to the expedited forwarding class. In *Proceedings of the 20th IEEE International Parallel Distributed Processing Symposium.*

[Martin et al., 2004] Martin, S., Minet, P., and George, L. (2004). The trajectory approach for the end-to-end response times with non-preemptive FP/EDF*. In *International Conference on Software Engineering Research and Applications*, Lecture Notes in Computer Science, pages 229–247. Springer, Berlin, Heidelberg.

[Martin et al., 2005] Martin, S., Minet, P., and George, L. (2005). End-to-end response time with fixed priority scheduling : trajectory approach versus holistic approach. *International Journal of Communication Systems*, 18(1) :37–56.

[Maxim and Cucu-Grosjean, 2013] Maxim, D. and Cucu-Grosjean, L. (2013). Response time analysis for fixed-priority tasks with multiple probabilistic parameters. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 224–235.

[Mayhew and Krishnan, 2003] Mayhew, D. and Krishnan, V. (2003). PCI express and advanced switching : evolutionary path to building next generation interconnects. In *11th Symposium on High Performance Interconnects*, pages 21–29.

[McKeown et al., 2008] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow : Enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2) :69–74.

[Mello et al., 2009] Mello, A., Calazans, N., and Moraes, F. (2009). QoS in networks-on-chip – beyond priority and circuit switching techniques. In *VLSI-SoC : Advanced Topics on Systems on a Chip*, IFIP International Federation for Information Processing, pages 1–22. Springer.

[Mifdaoui et al., 2007] Mifdaoui, A., Frances, F., and Fraboul, C. (2007). Real-time characteristics of switched ethernet for "1553b"-embedded applications : Simulation and analysis. In *2007 International Symposium on Industrial Embedded Systems*, pages 33–40.

[Mouftah and Sturgeon, 1990] Mouftah, H. and Sturgeon, R. (1990). Distributed discrete event simulation for communication networks. *IEEE Journal on Selected Areas in Communications*, 8(9) :1723–1734.

[Navet et al., 2000] Navet, N., Song, Y.-Q., and Simonot, F. (2000). Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over CAN (Controller Area Network). *Journal of Systems Architecture - The EUROMICRO Journal*, 46(7) :607–617.

[Nayak et al., 2016] Nayak, N. G., Dürr, F., and Rothermel, K. (2016). Time-sensitive Software-Defined Network (TSSDN) for Real-time Applications. In *24th International Conference on Real-Time and Network Systems, RTNS'16*.

[Nayak et al., 2017] Nayak, N. G., Dürr, F., and Rothermel, K. (2017). Routing Algorithms for IEEE 802.1Qbv Networks.

[Neukirchner et al., 2010] Neukirchner, M., Stein, S., Schrom, H., and Ernst, R. (2010). A software update service with self-protection capabilities. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 903–908.

[Neukirchner et al., 2011] Neukirchner, M., Stein, S., Schrom, H., Schlatow, J., and Ernst, R. (2011). Contract-based dynamic task management for mixed-criticality systems. In *6th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 18–27.

[Ni et al., 2011] Ni, M., Wu, X., Yu, Z., and Gao, B. (2011). A simple and fast algorithm for restricted shortest path problem. In *2011 Fourth International Joint Conference on Computational Sciences and Optimization*, pages 99–102.

[Noguero et al., 2012] Noguero, A., Calvo, I., and Almeida, L. (2012). A time-triggered middleware architecture for ubiquitous cyber physical system applications. *Proceedings of the 6th international conference on Ubiquitous Computing and Ambient Intelligence*, 7656 :73–80.

[Object Management Group, 2014] Object Management Group (2014). Data Distribution Service (DDS), Version 1.4.

[Oh et al., 2016] Oh, S., Lee, J., Lee, K., and Shin, I. (2016). RT-SDN : Adaptive routing and priority ordering for software-defined real-time networking.

[Osterlind et al., 2006] Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., and Voigt, T. (2006). Cross-level sensor network simulation with COOJA. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pages 641–648.

[Parkes, 2012] Parkes, S. (2012). SpaceWire User's Guide.

[PCI Special Interest Group, 2004] PCI Special Interest Group (2004). Conventional PCI 3.0 Specification.

[PCI Special Interest Group, 2014] PCI Special Interest Group (2014). PCI Express Base 3.1 Specification.

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

[Perumalla, 2006] Perumalla, K. (2006). Parallel and distributed simulation : Traditional techniques and recent advances. In *Proceedings of the 2006 Winter Simulation Conference*, pages 84–95.

[QEMU, 2017] QEMU (2017). QEMU 2.11.0 - https ://www.qemu.org.

[Quaglia, 2011] Quaglia, D. (2011). Timing aspects in QEMU/SystemC synchronization. *1st International QEMU Users' Forum*, 1 :11–14.

[Quereilhac et al., 2011] Quereilhac, A., Lacage, M., Freire, C., Turletti, T., and Dabbous, W. (2011). NEPI : An integration framework for network experimentation. In *2011 19th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–5.

[Quinton et al., 2012] Quinton, S., Ernst, R., Bertrand, D., and Yomsi, P. M. (2012). Challenges and new trends in probabilistic timing analysis. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 810–815.

[Raheja and Maakarn, 2014] Raheja, K. and Maakarn, S. (2014). A survey on different hybrid routing protocols of MANET. In *International Journal of Computer Science and Information Technologies (IJCSIT)*, volume 5, pages 5512–5516.

[Rakotoarivelo et al., 2010] Rakotoarivelo, T., Ott, M., Jourjon, G., and Seskar, I. (2010). OMF : A control and management framework for networking testbeds. *ACM SIGOPS Operating Systems Review*, 43(4) :54–59.

[Ramanathan, 1999] Ramanathan, P. (1999). Overload management in real-time control applications using (m,k)-firm guarantee. *IEEE Transactions on Parallel and Distributed Systems*, 10(6) :549–559.

[RapidIO Trade Association, 2014] RapidIO Trade Association (2014). RapidIO Specification 3.1.

[Rasche and Poize, 2005] Rasche, A. and Poize, A. (2005). Dynamic reconfiguration of component-based real-time software. In *10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005*, pages 347–354.

[Robert Bosch GmbH, 2011] Robert Bosch GmbH (2011). CAN with Flexible Data-Rate, Version 1.1.

[Rosen et al., 2001] Rosen, E., Viswanathan, A., and Callon, R. (2001). Multiprotocol Label Switching Architecture, RFC 3031.

[SAE International, 2011] SAE International (2011). AS6802 : Time-Triggered Ethernet.

[Scharbarg and Fraboul, 2007] Scharbarg, J. L. and Fraboul, C. (2007). Simulation for end-to-end delays distribution on a switched ethernet. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1092–1099.

[Scharbarg et al., 2009] Scharbarg, J. L., Ridouard, F., and Fraboul, C. (2009). A probabilistic analysis of end-to-end delays on an AFDX avionic network. *IEEE Transactions on Industrial Informatics*, 5(1) :38–49.

[Schmitt et al., 2003] Schmitt, J., Hurley, P., Hollick, M., and Steinmetz, R. (2003). Per-flow guarantees under class-based priority queueing. In *IEEE Global Telecommunications Conference, 2003 (GLOBECOM'03)*, volume 7, pages 4169–4174 vol.7.

[Schuster and Verma, 2008] Schuster, T. and Verma, D. (2008). Networking concepts comparison for avionics architecture. In *IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 1.D.1–1–1.D.1–11.

[Sebastian and Ernst, 2008] Sebastian, M. and Ernst, R. (2008). Modelling and designing reliable on-chip-communication devices in MPSoCs with real-time requirements. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 1465–1472.

[Sebastian and Ernst, 2009] Sebastian, M. and Ernst, R. (2009). Reliability analysis of single bus communication with real-time requirements. In *2009 15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 3–10.

[Shi and Burns, 2008] Shi, Z. and Burns, A. (2008). Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (NoCS 2008)*, pages 161–170.

[Shreedhar and Varghese, 1996] Shreedhar, M. and Varghese, G. (1996). Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3) :375–385.

[Silva et al., 2017] Silva, L., Gonçalves, P., Marau, R., and Pedreiras, P. (2017). Extending OpenFlow with industrial grade communication services. In *IEEE 13th International Workshop on Factory Communication Systems (WFCS)*, pages 1–4.

[Simonot et al., 2005] Simonot, F., Simonot-Lion, F., and Song, Y.-Q. (2005). Dependability Evaluation of Real-Time Applications Distributed on TDMA-Based Networks. IFAC.

[Sonkoly et al., 2012] Sonkoly, B., Gulyás, A., Németh, F., Czentye, J., Kurucz, K., Novák, B., and Vaszkun, G. (2012). On QoS support to ofelia and OpenFlow. In *2012 European Workshop on Software Defined Networking*, pages 109–113.

[Specht and Samii, 2017] Specht, J. and Samii, S. (2017). Performance comparison of IEEE 802.1Q and IEEE 802.1 AVB in an Ethernet-based in-vehicle network.

[Strassburger et al., 2008] Strassburger, S., Schulze, T., and Fujimoto, R. (2008). Future trends in distributed simulation and distributed virtual environments : Results of a peer study. In *Proceedings of the 2008 Winter Simulation Conference*, pages 777–785.

[Taylor, 2003] Taylor, S. (2003). *HLA-CSPIF The High Level Architecture COTS Simulation Package Interoperability Forum.*

[Taylor et al., 2006] Taylor, S., Wang, X., Turner, S., and Low, M. (2006). Integrating heterogeneous distributed COTS discrete-event simulation packages : an emerging standards-based approach. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans*, 36(1) :109–122.

[Ternon et al., 2016] Ternon, C., Goossens, J., and Dricot, J.-M. (2016). FFT-OpenFlow, on the way towards real-time SDN. In *Proceedings of the 14th International Workshop on Real-Time Networks.*

[Thiele and Ernst, 2016] Thiele, D. and Ernst, R. (2016). Formal worst-case performance analysis of time-sensitive ethernet with frame preemption. In *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9.

[Tindell and Clark, 1994] Tindell, K. and Clark, J. (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2) :117–134.

[Tobuschat et al., 2013] Tobuschat, S., Axer, P., Ernst, R., and Diemer, J. (2013). IDAMC : A NoC for mixed criticality systems. In *IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 149–156.

[TSN Task Group, 2011] TSN Task Group (2011). IEEE 802.1BA : Audio Video Bridging (AVB) Systems.

[TSN Task Group, 2016a] TSN Task Group (2016a). IEEE 802.1AS : Timing and Synchronization for Time-Sensitive Applications (gPTP).

[TSN Task Group, 2016b] TSN Task Group (2016b). IEEE 802.1Qav : Forwarding and Queuing for Time-Sensitive Streams.

[TSN Task Group, 2016c] TSN Task Group (2016c). IEEE 802.1Qbv : Enhancements for Scheduled Traffic.

[TSN Task Group, 2016d] TSN Task Group (2016d). IEEE 802.1Qca : Path Control and Reservation.

[TSN Task Group, 2016e] TSN Task Group (2016e). IEEE 802.1Qcb : Seamless Redundancy.

[TSN Task Group, 2016f] TSN Task Group (2016f). IEEE 802.1Qcc : Stream Reservation Protocol (SRP) Enhancements and Performance Improvements.

[UC Berkeley, 2014] UC Berkeley (2014). Ptolemy II 10.0 - http ://ptolemy.eecs.berkeley.edu/ptolemyII.

[US DoD, 1978] US DoD (1978). CMIL-STD-1553B : Digital Time Division Command/Response Multiplex Data Bus.

[Van Mieghem and Kuipers, 2003] Van Mieghem, P. and Kuipers, F. A. (2003). On the complexity of QoS routing. *Computer Communications*, 26 :376–387.

[Vaubourg et al., 2015] Vaubourg, J., Presse, Y., Camus, B., Bourjot, C., Ciarletta, L., Chevrier, V., Tavella, J.-P., and Morais, H. (2015). Multi-agent multi-model simulation of smart grids in the MS4sg project. In *Advances in Practical Applications of Agents, Multi-Agent Systems, and Sustainability : The PAAMS Collection*, volume 9086, pages 240–251. Springer International Publishing.

[Vestal, 2007] Vestal, S. (2007). Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE International Real-Time Systems Symposium*, pages 239–243.

[Wang and Crowcroft, 1995] Wang, Z. and Crowcroft, J. (1995). Bandwidth-delay based routing algorithms. In *IEEE Global Telecommunications Conference (GLOBECOM'95)*, volume 3, pages 2129–2133 vol.3.

[White et al., 2002] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. *ACM SIGOPS Operating Systems Review*, 36 :255–270.

[Wilwert et al., 2003] Wilwert, C., Song, Y., Simonot-Lion, F., Loria-Trio, and Clement, T. (2003). Evaluating quality of service and behavioral reliability of steer-by-wire systems. In *Proceedings of the 2003 IEEE Conference on Emerging Technologies and Factory Automation*, volume 1, pages 193–200 vol.1.

[Wu et al., 2016] Wu, Y., Lu, C., and Chen, Y. (2016). A survey of routing algorithm for mesh network-on-chip. In *Frontiers of Computer Science*, volume 10, pages 591–601.

[Xu et al., 2015] Xu, C., Chen, B., and Qian, H. (2015). Quality of service guaranteed resource management dynamically in software defined network. *Journal of Communications*, 10 :843–850.

[Xu and Parnas, 1993] Xu, J. and Parnas, D. (1993). On satisfying timing constraints in hard-real-time systems. *IEEE Transactions on Software Engineering*, 19(1) :70–84.

[Yen, 1971] Yen, J. Y. (1971). Finding the K Shortest Loopless Paths in a Network. *Management Science*, 17(11) :712–716.

[Yomsi and Sorel, 2007] Yomsi, P. M. and Sorel, Y. (2007). Extending Rate Monotonic Analysis with Exact Cost of Preemptions for Hard Real-Time Systems. In *Proceedings of 19th Euromicro Conference on Real-Time Systems, ECRTS'07*.

[Zeigler et al., 2000] Zeigler, B. P., Kim, T. G., and Praehofer, H. (2000). *Theory of Modeling and Simulation*. Academic Press, Inc., 2nd edition.

[Zhang et al., 1997] Zhang, L., Berson, S., Herzog, S., and Jamin, S. (1997). Resource ReSerVation protocol (RSVP) – version 1 functional specification, RFC 2205.

[Zhou et al., 2011] Zhou, Q., Qu, Z., and Lin, H. (2011). Admission control of VL in AFDX under HRT constraints. *Chinese Journal of Aeronautics*, 24(2) :195–201.

[Zirin and Bennett, 2004] Zirin, S. and Bennett, J. (2004). Advanced Switching Overview.

Annexe A

Interfaces de programmation SDRN

Ce document, rédigé par Romain Tissier, stagiaire ingénieur à Thales Research & Technology, récapitule les interfaces de programmation SDRN : interface utilisateur, interface de configuration, interface de supervision et interface de requête de flux.

Présentation de l'API SDRN

Romain TISSIER

6 avril 2017

1 Introduction

Dans le cadre de la mise en œuvre d'une interface applicative haut niveau pour l'utilisation d'un contrôleur SDRN[1], nous avons spécifié plusieurs interfaces de programmation à différent niveaux. Ce document va donc décrire l'architecture cible et les hypothèses associées avant de présenter de manière détaillée les interfaces:

- Utilisateur
- Applicative
- Configuration interne
- Configuration externe
- Supervision

2 Architecture cible

Dans la configuration souhaitée, l'application utilisateur utilisera l'API socket, qui appellera le driver SDRN selon ses besoins (figure 1). Ce driver SDRN embarquera les primitives nécessaires à la mise en œuvre de la communication sur le réseau SDRN. C'est à dire, au minimum l'interface applicative et l'interface de configuration interne.

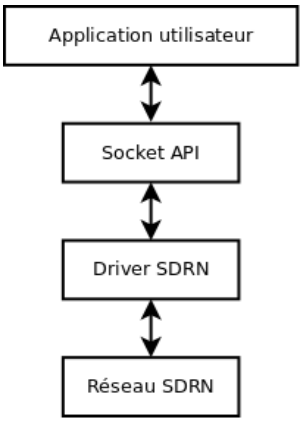

Figure 1: Architecture des interfaces de programmation

[1]Software-Defined Real-Time Networking

3 Hypothèses

Pour faciliter la mise en œuvre du réseau SDRN, nous utiliseront deux hypothèses :

- Nous considérerons que les applications sont fixes sur le réseau maillé. C'est à dire qu'elles connaissent les ports utilisés par les applications distantes (en contradiction avec un fonctionnement où l'on demanderait à un serveur d'ouvrir un port).
- Dans un premier temps, l'allocation d'un flux ne pourra pas se faire en utilisant la fonction `connect` (décrite ci-dessous). Il faudra donc réaliser une étape supplémentaire qui précisera manuellement les propriétés du flux à réserver que nous matérialiseront par la fonction `connect_flow`. Plus tard, cette information sera connue par le contrôleur.

4 Interface utilisateur

Pour l'interface avec l'utilisateur, il a été décidé d'utiliser l'interface de programmation standard fournie par le système Linux: les sockets. Ainsi l'application utilisateur pourra utiliser le réseau SDRN de la manière la plus transparente possible. Nous allons donc décrire cette API et présenter les moyens adaptatifs mis en œuvre pour prendre en charge le SDRN.

4.1 Description de l'API socket

L'API socket propose des fonctions génériques permettant de mettre en œuvre une communication entre deux applications distantes. Ces fonctions sont détaillées ci-dessous.

4.1.1 Création d'une socket

La création d'une socket est réalisée par la fonction :

```
int socket(int domain, int type,
           int protocol);
```

Cette fonction prend en premier argument le domaine de communication. Généralement le domaine utilisé est `AF_INET`. Ce domaine correspond à l'utilisation de l'IP. Cependant, Linux propose d'autres protocoles qui sont codés directement dans le noyau.

Le second argument de cette fonction est le type de socket. On peut utiliser plusieurs types de socket, et plus particulièrement :

- `SOCK_STREAM`, qui correspond à un mode connecté de transmission de flux, il nécessite une connexion préalable mais il permet de fiabiliser la communication. Ce mode est par exemple utilisé par le protocole TCP/IP.
- `SOCK_DGRAM`, qui correspond à un mode déconnecté de transmission de datagramme, il ne nécessite pas de connexion mais ne permet pas seul d'assurer la fiabilité de la transmission. Ce mode est par exemple utilisé par le protocole UDP/IP.
- `SOCK_RAW`, qui correspond à une socket vide configurable manuellement. Ce type est par exemple utilisé pour manipuler directement le protocole ICMP[2].

Le dernier argument permet de spécifier le protocole utilisé par la socket. La liste des protocoles pris en charge par le système, ainsi que les identifiants associés sont répertoriés dans le fichier `/etc/protocols`.

Enfin, la fonction `socket` retourne le descripteur correspondant à la socket créée ou -1 en cas d'erreur.

4.1.2 Nommage de la socket

Une fois créée, il est nécessaire de lier une adresse à la socket. On appelle cette étape le nommage. Pour cela, il faut utiliser la fonction :

```
int bind(int sockfd, const struct sockaddr
        *addr, socklen_t addrlen);
```

Cette fonction prend en argument le descripteur de la socket, l'adresse de la socket au format de la structure `sockaddr` et la taille de cette structure.

`sockaddr` est composé de la famille d'adresse et de l'adresse elle-même :

```
struct sockaddr {
    unsigned short      sa_familydf;
    char                sa_data[14];
};
```

Pour faciliter l'utilisation d'autre types d'adresses, on peut caster cette structure. Ce procédé est directement utilisé dans le code du noyau pour la prise en charge des adresses IPv4 et IPv6.

Enfin, cette fonction renvoie 0 ou -1 en cas d'erreur.

4.1.3 Se connecter à l'application distante

Si on opère dans un mode connecté, il est nécessaire de se connecter au socket de l'application distante. Pour cela, on utilisera la fonction :

```
int connect(int sockfd, const struct
        sockaddr *addr, socklen_t addrlen);
```

[2]Internet Control Message Protocol

Cette fonction prend en argument le descripteur de la socket, l'adresse de la socket destinataire ainsi que sa taille. Elle retourne 0 ou -1 en cas d'erreur.

4.1.4 Envoyer un message

Pour envoyer un message, en mode connecté, il faut utiliser la fonction :

```
ssize_t send(int sockfd, const void *buf,
        size_t len, int flags);
```

Ou bien en mode déconnecté, la fonction :

```
ssize_t sendto(int sockfd, const void *buf,
        size_t len, int flags, const struct
        sockaddr *dest_addr, socklen_t
        addrlen);
```

Ces fonctions prennent en argument le descripteur de la socket, la donnée à écrire, sa taille, des drapeaux et si nécessaire l'adresse à laquelle envoyer la donnée.

Elles retournent le nombre de caractères envoyés ou -1 en cas d'erreur.

4.1.5 Recevoir un message

Pour recevoir un message en mode connecté, il faut utiliser la fonction:

```
ssize_t recv(int sockfd, void *buf, size_t
        len, int flags);
```

Ou bien en mode déconnecté, la fonction :

```
ssize_t recvfrom(int sockfd, void *buf,
        size_t len, int flags, struct
        sockaddr *src_addr, socklen_t
        *addrlen);
```

Ces fonctions prennent en argument le descripteur de la socket, la donnée à envoyer ainsi que sa taille, des drapeaux et si nécessaire l'adresse de l'émetteur du message.

Elles retournent le nombre de caractères reçus ou -1 en cas d'erreur.

4.1.6 Fermer la socket

Pour fermer la communication proprement et éviter les fuites mémoires, il est nécessaire de fermer la socket avec la fonction:

```
int close(int fd);
```

Cette fonction permet simplement de clore le descripteur de fichier passé en argument. Elle renvoie 0 ou -1 s'il y a une erreur.

4.2 Adaptation

Pour prendre en charge le contrôleur SDRN, il sera nécessaire de réaliser des adaptations sur cette API socket.

4.2.1 Type de socket

Nous utiliserons un mode datagrame fiable pour les transmissions temps réel. Nous utiliserons donc des sockets de type `SOCK_DGRAM` avec un dimensionnement du réseau permettant d'assurer la fiabilité des transmissions.

4.2.2 Type d'adresse

Il est nécessaire de modifier la famille d'adresse utilisé par la socket. En effet, comme nous n'utiliseront pas le mode IP, nous ne pourrons pas utiliser le type `AF_INET` utilisé classiquement. Pour cela, il semblerait que nous pourrions utiliser le drapeaux `AF_UNSPEC` qui correspond à un adressage non spécifié. Sinon, il faudra peut être adapter le code du noyau (situé dans net/) pour créer cet adressage. Une autre solution sera peut être d'utiliser le drapeau `AF_PACKET` qui permettrait d'utiliser un système de communication par paquet basique.

4.2.3 Structure d'adresse

En lien avec le nouveau type d'adresse, nous devons modifier la structure associée, pour cela nous utiliserons notre propre structure qui contiendra :

- La priorité du flux (sur 3 bits)
- L'identifiant du flux (mode connecté) ou identifiant du destinataire (mode déconnecté) (sur 16 bits)
- Le port d'écoute du destinataire (sur 16 bits)
- Le type de flux (sur 2 bits)

Cette structure sera ensuite castée en `sockaddr` pour être utilisée par l'API socket.

Lorsque nous utiliseront la fonction `connect`, cette structure d'adresse sera utilisée pour spécifier l'identifiant de flux. Ainsi la structure passée en paramètre sera modifiée directement à l'intérieur de la fonction.

4.2.4 Utilisation des priorités

Le réseau SDRN permet de passer des messages en "Best-Effort". Pour utiliser ce mode, il n'est pas nécessaire d'allouer de flux via la fonction `connect`. Il suffit de directement utiliser la fonction `sendto`. Cependant le réseau propose d'utiliser un système de priorité pour ces flux en "Best-Effort". C'est pourquoi nous pensons renseigner cette priorité grâce à un drapeau directement passé à la fonction `sendto`. En effet, en TCP, l'utilisateur pourra utiliser le flag "URG" pour signaler un message prioritaire. Il aura au sein du réseaux une priorité 7, alors que les autres flux auront une priorité 8 par défaut.

4.3 Interfaçage

Nous avons proposé des primitives prédéfinies pour utilisé le réseau SDRN. Celle-ci seront temporairement encapsulées dans les fonctions de l'API socket. Ainsi :

- `request` sera appelé par la fonction `connect_flow`
- `accept` sera appelé par la fonction `connect_flow`
- `refuse` sera appelé par la fonction `connect_flow`
- `release` sera appelé par la fonction `close`

5 Interface applicative

L'interface applicative est pour l'instant située sous l'API socket. Elle est encapsulée. Cette interface propose les structures et les fonctions:

```
typedef enum {
        PERIODIC_REAL_TIME,
        NON_REAL_TIME
} FlowType;

typedef struct {
        int id;
        FlowType type;
        double period;
        int maximal_data_bundle_size;
        double deadline;
        int source;
        int destination;
        int *path;
        int path_lenght;
        int external_link_weight;
        int internal_link_weight;
} Flow;

typedef struct {
        FlowType type;
        double period;
        int maximal_data_bundle_size;
        double deadline;
        int source;
        int destination;
} FlowProperties;

Flow* request(FlowProperties properties);
int accept(int flow_id);
int refuse(int flow_id);
int release(int flow_id);
```

Ainsi la fonction `request` permet de demander au contrôleur l'allocation d'un flux selon les contraintes passées en argument. Si le contrôleur a la capacité de respecter ces contraintes, il renvoie une structure contenant les informations du flux allouable.

Selon les informations contenues dans la structure `Flow`, on peut accepter (`accept`) ce flux ou le refuser (`refuse`). La première solution l'alloue alors que la seconde libère l'espace réservé pour ce flux.

Quand le flux a été accepté mais n'est plus utilisé, on peut le relâcher grâce à la fonction `release`.

6 Interface de configuration interne

La configuration interne permet de configurer le switch local. Les primitives utilisées seront :

```
int add_flow(int flow_id, int weight,
        int egress_id[4], int recovery);
int release_flow(int flow_id);
int add_route(int destination_id,
        int egress_id[4]);
int clear_route(int destination_id);
int add_ip_mapping(char dest_ip[16],
        int flow_id, int dest_id);
int remove_ip_mapping(char ip[16]);
```

On pourra ainsi via cette interface ajouter un flux, en précisant ses propriétés et le relâcher. On pourra aussi ajouter un chemin vers un nœud et supprimer tout les chemins qui mènent à un nœud. Ces chemins sont passé dans le tableau `egress_id` dans leur ordre de priorité (le premier élément du tableau possède la plus grande priorité et un chemin nul équivaux au port -1). Lors de l'ajout d'un flux, on précise aussi s'il peut ou non utiliser le mode "recovery" en renseignant le booléen `recovery`.

Les fonctions de gestions du mappage entre les applications(qui sont référencée grâce à une ip unique) et les identifiants de flux, permettent de gérer une table locale qui ajoutera directement dans la trame émise par l'application l'identifiant de flux associé.

Toutes les fonctions d'ajout permettent si l'élément est déjà en mémoire d'écraser la valeur précédente.

7 Interface de configuration externe

L'interface de configuration externe est l'interface permettant de communiquer avec le logiciel SCADE dans le cadre du projet S3P/WP7. Elle propose les primitives java :

```
int configureFlow(int flowId, int externWeight,
        int internWeight, List<String> path);
int configureFlow(Flow flow);
int clearFlow(int flowId);
```

Ainsi elle permet de configurer un flux de bout en bout et de supprimer un flux.

8 Interface de supervision

Toujours dans le cadre du projet S3P/WP7, l'interface de supervision permet d'administrer le réseau grâce aux primitives java:

```
void nodeFailure(String nodeId);
int nodeRestart(String nodeId);
void linkFailure(String node1, String node2);
int linkRestart(String node1, String node2);
```

Ainsi elle gère les erreurs sur les nœuds ou les liens et peux redémarrer un lien ou un nœud.

9 Sources

- Florian Greff, Ye-Qiong Song, Laurent Ciarletta, Arnaud Samama. *A dynamic Flow Allocation Method to the Design of a Software-Defined Real-Time Mesh Network*, 2017
- Le manuel Linux
- Le code source du noyau linux (net/)

Annexe B

Architecture et utilisation du contrôleur

Cette annexe décrit l'architecture logicielle du contrôleur SDRN Java et récapitule son utilisation.

B.1 Paramètres à fournir en entrée

B.1.1 Préliminaires

— N_i définit le noeud de numéro i.
— L définit un lien. LE définit un lien externe (d'un noeud à un autre), identiques entre eux. LI définit un lien interne (entre la tuile de calcul et le noeud ou l'inverse), identiques entre eux.

B.1.2 Paramètres de la plateforme

— $d_{sw,\mathcal{R}}$: latence de commutation, constant dans tout le réseau.
— Représentation de la topologie (et donc de l'état du réseau) sous la forme d'une liste d'adjacence. Par exemple, la liste {1-2, 2-1, 1-3} définit un réseau constitué de 3 noeuds, d'un lien bidirectionnel entre N_1 et N_2 et d'un lien unidirectionnel de N_1 vers N_3. Les liens internes sont implicites et ne sont pas représentés sur ce graphe.

B.1.3 Paramètres des liens externes

— C_{LE}, capacité (débit) maximale ;
— η_{LE}, taille maximale d'entête du protocole sous-jacent ;
— ψ_{LE}, taille de payload maximale du protocole sous-jacent ;
— Ch_{LE}, nombre total de sous-canaux présents sur les liens ;
— Q_{LE}, quantum. Il s'agit du crédit maximal alloué à chaque sous-canal à chaque tour du jeton.

Les deux derniers paramètres sont liés au *Credit-Based Round Robin*, qui est l'ordonnancement utilisé en sortie de noeud. On peut supposer que le CBRR existe sur la plateforme et définir Ch_{LE} et Q_{LE} arbitrairement.

B.1.4 Paramètres des liens internes

De la même façon que ci-dessus, on définit C_{LI}, η_{LI}, ψ_{LI}, Ch_{LI}, Q_{LI}.

B.1.5 Paramètres des requêtes et flux

On fournit en entrée du contrôleur la requête du flux f, composée des valeurs suivantes :
— $type_f$, le type du flux : soit Periodic Real-Time (PRT), soit Non Real-Time (NRT). Les valeurs ci-après ne sont pour le moment utilisées que pour les flux PRT ;
— T_f, période minimale du flux. Intervalle de temps minimal entre deux envois de données par l'application ;
— S_f, taille maximale des données brutes (*payload* pure) à chaque envoi ;
— dl_req_f, deadline réseau relative ;
— src_f, noeud source ;
— $dest_f$, noeud destination.

Le contrôleur SDRN étudie cette requête qui sera soit rejetée soit acceptée. On peut envisager de préciser les causes du rejet en fonction des besoins de la couche supérieure.

Si le flux est accepté, le contrôleur SDRN est capable de retourner les informations suivantes :
— id_f, ID du flux, généré automatiquement. Il doit être placé dans l'entête à chaque envoi de données ;
— $path_f$, chemin suivi par le flux en mode nominal. Sera transcris en configurations de tables de routage. Le format du chemin est la liste des noeuds traversés ;
— $w_{f,LE}$, poids du flux sur les liens externes. Utilisé par le *Credit-Based Round Robin*. Sera transcris en configurations sur les noeuds traversés par le flux ;
— $w_{f,LI}$, poids du flux sur les liens internes. Cf. ci-dessus.

B.2 Implémentation en Java

B.2.1 Diagramme de classe

Voir Figure B.1.

B.2.2 Modèle du réseau

Les nœuds sont représentés par des chaînes de caractères. Ils pourraient être encapsulés dans une classe *Node* à l'avenir pour une meilleur cohérence architecturale.

Le contrôleur SDRN travaille sur un objet *Network* représentant le réseau sous la forme d'une liste d'adjacence. Pour créer ce réseau, on instancie un réseau vide en indiquant ses paramètres (cf. section précédente), puis on ajoute ou retire des liens avec les fonctions *addEdge(String, String)* et *removeEdge(String, String)*. Le reste est géré de façon transparente.

Je propose également deux fonctions pour instancier automatiquement des réseaux toriques ou maillés de dimension n : *generateTorusNetwork(n)* et *generateMeshNetwork(n)*.

Ces fonctions pourront évoluer en fonction de vos besoins (par exemple instancier l'objet à partir d'un fichier externe). Let me know.

B.2.3 Modèle des flux et requêtes

L'objet *Properties* contient les paramètres de requête du flux : T_f, S_f, dl_req_f, src_f, $dest_f$. Il faudrait ajouter $type_f$ mais je ne l'utilise pas pour le moment car je considère qu'on n'alloue que des flux PRT.

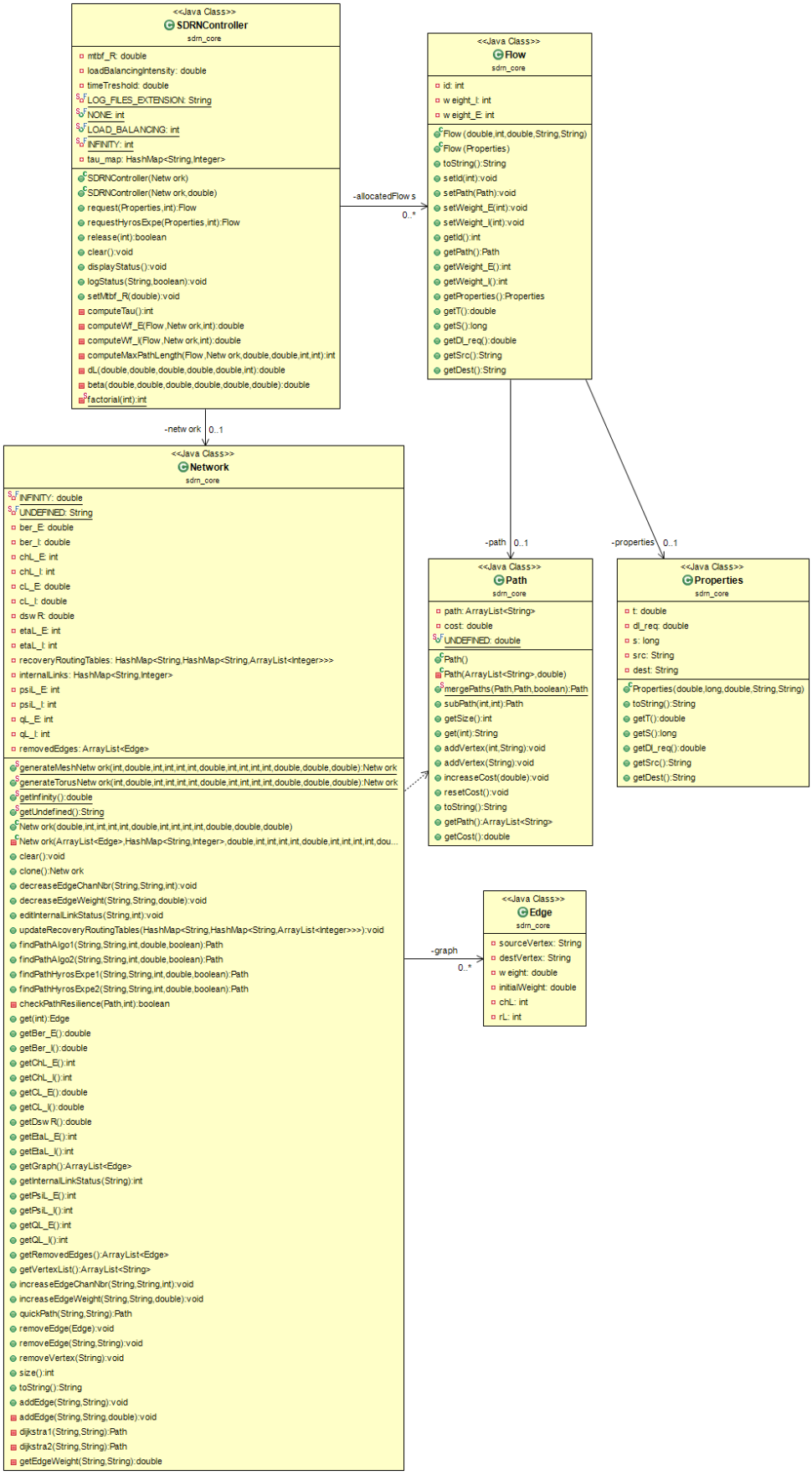

FIGURE B.1 – Modèle UML du contrôleur SDRN

L'objet *Flow* contient les propriétés ci-dessus, plus les propriétés résultant du contrôle d'admission : id_f, $path_f$, $w_{f,LE}$, $w_{f,LI}$. Le chemin $path_f$ est représenté par un objet *Path*, possédant un attribut *path* correspondant à la liste des noeuds traversés.

B.3 Utilisation du contrôleur

B.3.1 Utilisation générale

On instancie un contrôleur SDRN en lui fournissant l'objet *Network*. Ensuite, on effectue des requêtes de flux via la fonction *request(Properties)*. Cette fonction retourne soit *null* si le flux est rejeté, soit l'objet *Flow* rempli.

On peut libérer les ressources d'un flux grâce à la fonction *release(id du flux)*.

B.3.2 Utilisation dans le projet S3P/WP7

Interface Scade vers SDRN Controller : configuration

`int configureFlow (Flow f)`
Lors du calcul des configurations, le contrôleur SDRN va retourner l'ensemble des Flows correspondant à cette configuration. Ces objets Flow peuvent être directement utilisés en paramètres de la méthode. Retourne un éventuel code d'erreur ou le succès.

`int configureFlow (int flowId, int wfe, int wfi, ArrayList<String> path)`
Si l'utilisation des objets Flow pose problème, on peut directement utiliser ces paramètres plus génériques. Ils sont récupérables depuis les objets Flow. Retourne un éventuel code d'erreur ou le succès.

`int clearFlow (int flowId)`
Supprime l'allocation du flux de numéro flowId. Retourne un éventuel code d'erreur ou le succès.

Interface SDRN Controller vers Scade : supervision

`void nodeFailure (String nodeId)`
Indique que le noeud d'identifiant nodeId est tombé en panne.

`void linkFailure (String node1, String node2)`
Indique que le lien reliant les noeuds d'identifiants node1 et node2 est tombé en panne. L'ordre n'a aucune importance car le lien est tombé dans les deux sens.

`int nodeRestart (String nodeId)`
Indique que le noeud d'identifiant nodeId, précédemment tombé en panne, est à nouveau fonctionnel. Retourne un éventuel code d'erreur ou le succès.

`int linkRestart (String node1, String node2)`
Indique que le lien reliant les noeuds d'identifiants node1 et node2, précédemment tombé en panne, est à nouveau fonctionnel. L'ordre des paramètres n'a pas d'importance. Retourne un éventuel code d'erreur ou le succès.

Résumé / Abstract

Dans le cadre d'une thèse CIFRE conjointe entre le Loria et Thales Research & Technology, nous étudions un nouveau type de réseau maillé embarqué temps réel. La mise en réseau maillé des composants des systèmes embarqués concilie les contraintes temps réel des applications avec leurs besoins croissants en termes de bande passante et collaboration. La pluralité des chemins de communication résulte en de meilleures propriétés de flexibilité, résilience, passage à l'échelle et répartition de charge. Cependant, ceci nécessite d'être capable d'allouer dynamiquement les ressources réseau en fonction des besoins des applications. Notre approche consiste à permettre aux applications de faire des requêtes de flux temps réel à l'exécution, puis allouer dynamiquement les ressources correspondant aux besoins en communication. A cette fin, nous avons conçu l'architecture Software-Defined Real-time Networking (SDRN). Elle aborde en même temps les problématiques d'isolation des flux, analyse temporelle, routage, tolérance aux fautes, ainsi que les interfaces avec les couches applicatives et les couches basses du système. Elle est également modulaire, c'est-à-dire que certaines parties de l'architecture peuvent être remplacées sans remettre en cause les autres modules. Enfin, elle a été validée par une implémentation sur plateforme matérielle RapidIO. Ce document restitue les travaux de recherche sur SDRN. Il s'intéresse également à la problématique de l'expérimentation sur les réseaux embarqués et propose une approche originale d'expérimentation, ERICA. Cette approche facilite la mise en place d'expérimentations mêlant aspects réels et simulés. ERICA génère les fichiers nécessaires à la mise en place du scénario défini dans une interface graphique haut niveau. Elle permet ainsi au chercheur d'appliquer une réflexion haut niveau sur ses expérimentations et de réutiliser les couches communes à plusieurs scénarios d'expérimentation.

Mots-clés : réseau, temps réel, embarqué, SDN, maillé

We are studying a new kind of embedded real-time mesh network. Mesh networking of the components of embedded systems reconciles their real-time constraints with the new application needs in terms of bandwidth and tight interactions. The plurality of communication paths results in increased flexibility, resilience, scalability and load balancing characteristics. However, this requires the ability to dynamically allocate network resource with respect to the needs of running applications. Our approach is to allow applications to make online real-time flow resource requests and consequently allot network resources according to these requirements. To this end, we have designed the Software-Defined Real-time Networking (SDRN) architecture. It addresses flow isolation, timing analysis, routing, fault tolerance, as well as the interfaces with the application layer and the lower layers of the system. It also allows any module to be replaced without interfering with the remainder of the architecture. It has been validated via an implementation on an in-silicon RapidIO platform. This thesis describes our research on the SDRN architecture. It also proposes an original method for experimenting on embedded networks, ERICA. The ERICA framework automatically generates all what is needed to conduct a network experiment in a selected environment (such as a simulator or a testbed), with both physical and simulated aspects. Hence, it allows the researcher to perform a high-level thinking of the whole experimentation process and to reuse applications and experiment designs from an experimentation stack to another.

Keywords : network, real-time, embedded, SDN, mesh