Proposition et vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil

Alexandre Mouradian

Numéro d'ordre XXXXXXXX Année 2013

Thèse

Proposition et vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil

Thèse présentée devant
L'Institut National des Sciences Appliquées de Lyon

pour obtenir
le grade de docteur

Ecole doctorale : Informatique et Mathématiques de Lyon

par
Alexandre Mouradian

Soutenue le 18 novembre 2013 devant la commission d'examen

Jury

Christian Fraboul	Professeur des Universités INP - ENSEEIHT	Rapporteur
Congduc Pham	Professeur des Universités Laboratoire LIUPPA - Université de Pau	Rapporteur
Vania Conan	Directeur de recherche Thales Communications & Security	Examinateur
Michel Misson	Professeur des Universités Laboratoire LIMOS - ISIMA	Examinateur
Jean-Marc Thiriet	Professeur des Universités Laboratoire GIPSA - UJF	Examinateur
Isabelle Augé-Blum	Maître de Conférences Laboratoire CITI - INSA	Encadrante
Fabrice Valois	Professeur des Universités Laboratoire CITI - INSA	Directeur

Les travaux présentés dans ce mémoire ont été réalisés au laboratoire CITI sous la direction de Dr. Isabelle Augé-Blum et Pr. Fabrice Valois, et partiellement financé par le Ministère de la Recherche sous le contrat : ARESA2 ANR-09-VERS-017.

Résumé

Nos travaux portent sur les protocoles de communications temps-réel pour Réseaux de Capteurs sans Fil (RCsF). Les RCsF sont des réseaux ad hoc, sans fil, large échelle déployés pour mesurer des paramètres de l'environnement et remonter les informations à un ou plusieurs emplacements (nommés puits). Les éléments qui composent le réseau sont de petits équipements électroniques qui ont de faibles capacités en termes de mémoire et de calcul ; et fonctionnent sur batterie. Ces caractéristiques font que les protocoles développés, dans la littérature scientifique de ces dernières années, visent principalement à auto-organiser le réseau et à réduire la consommation d'énergie. Avec l'apparition d'applications critiques pour les réseaux de capteurs sans fil, de nouveau besoins émergent, comme le respect de bornes temporelles et de fiabilité. En effet, les applications critiques sont des applications dont dépendent des vies humaines ou l'environnement, un mauvais fonctionnement peut donc avoir des conséquences catastrophiques. Nous nous intéressons spécifiquement aux applications de détection d'événements et à la remontée d'alarmes (détection de feu de forêt, de glissement de terrain, d'intrusion, etc), ces applications ont des contraintes temporelles strictes.

D'une part, dans la littérature, on trouve peu de protocoles qui permettent d'assurer des délais de bout en bout bornés. Parmi les propositions, on trouve des protocoles qui permettent effectivement de respecter des contraintes temporelles mais qui ne prennent pas en compte les spécificités des RCsF (énergie, large échelle, etc). D'autres propositions prennent en compte ces aspects, mais ne permettent pas de garantir des bornes temporelles. D'autre part, les applications critiques nécessitent un niveau de confiance très élevé, dans ce contexte les tests et simulations ne suffisent pas, il faut être capable de fournir des preuves formelles du respect des spécifications. A notre connaissance cet aspect est très peu étudié pour les RcsF. Nos contributions sont donc de deux types :

- Nous proposons un protocole de remontée d'alarmes, en temps borné, X-layer (MAC/routage, dénommé RTXP) basé sur un système de coordonnées virtuelles originales permettant de discriminer le 2-voisinage. L'exploitation de ces coordonnées permet d'introduire du déterminisme et de construire un gradient visant à contraindre le nombre maximum de sauts depuis toute source vers le puits. Nous proposons par ailleurs un mécanisme d'agrégation temps-réel des alarmes remontées pour lutter contre les tempêtes de détection qui entraînent congestion et collision, et donc limitent la fiabilité du système.
- Nous proposons une méthodologie de vérification formelle basée sur les techniques de Model Checking. Cette méthodologie se déroule en trois points, qui visent à (1) modéliser de manière efficace la nature diffusante des réseaux sans fil, (2) vérifier les RCsF en prenant en compte la non-fiabilité du lien radio et (3) permettre le passage à l'échelle de la vérification en mixant *Network Calculus* et *Model Checking*. Nous appliquons ensuite cette méthodologie pour vérifier RTXP.

Abstract

Wireless Sensor Networks (WSNs) are ad hoc wireless large scale networks deployed in order to monitor physical parameters of the environment and report the measurements to one or more nodes of the network (called sinks). The small electronic devices which compose the network have low computing and memory capacities and run on batteries, researches in this field have thus focused mostly on self-organization and energy consumption reduction aspects. Nevertheless, critical applications for WSNs are emerging and require more than those aspects, they have real-time and reliability requirements. Critical applications are applications on which depend human lives and the environment, a failure of a critical application can thus have dramatic consequences. We are especially interested in anomaly detection applications (forest fire detection, landslide detection, intrusion detection, etc), which require bounded end to end delays and high delivery ratio.

Few WSNs protocols of the literature allow to bound end to end delays. Among the proposed solutions, some allow to effectively bound the end to end delays, but do not take into account the characteristics of WSNs (limited energy, large scale, etc). Others, take into account those aspects, but do not give strict guaranties on the end to end delays. Moreover, critical applications require a very high confidence level, simulations and tests are not sufficient in this context, formal proofs of compliance with the specifications of the application have to be provided. The application of formal methods to WSNs is still an open problem.

Our contributions are thus twofold :

- We propose a real-time cross-layer protocol for WSNs (named RTXP) based on a virtual coordinate system which allows to discriminate nodes in a 2-hop neighborhood. Thanks to these coordinates it is possible to introduce determinism in the accesses to the medium and to bound the hop-count, this allows to bound the end to end delay. Besides, we propose a real-time aggregation scheme to mitigate the alarm storm problem which causes collisions and congestion and thus limit the network lifetime.
- We propose a formal verification methodology based on the Model Checking technique. This methodology is composed of three elements, (1) an efficient modeling of the broadcast nature of wireless networks, (2) a verification technique which takes into account the unreliability of the wireless link and (3) a verification technique which mixes Network Calculus and Model Checking in order to be both scalable and exhaustive. We apply this methodology in order to formally verify our proposition, RTXP.

Remerciements

Je tiens tout d'abord à remercier mon encadrante de thèse Isabelle Augé-Blum, Maître de Conférence à l'INSA de Lyon, pour m'avoir donné l'opportunité de réaliser cette thèse, d'avoir été toujours présente pour me former et m'orienter durant ces trois années. Je tiens aussi à remercier mon directeur de thèse Fabrice Valois, Professeur des Universités à L'INSA de Lyon, pour ses conseils et les discussions scientifiques partagées au cours de ma thèse.

Je remercie Christian Fraboul, Professeur des Universités à l'ENSEEIHT et Congduc Pham, Professeur des Universités au Laboratoire LIUPPA d'avoir accepté d'être rapporteurs de cette thèse. Je remercie également Vania Conan, Directeur de recherche chez Thales Communications & Security, Michel Misson, Professeur des Universités au Laboratoire LIMOS, Jean-Marc Thiriet, Professeur des Universités au Laboratoire GIPSA d'avoir accepté d'évaluer mon travail.

Ce travail a été réalisé au sein de l'équipe INRIA Urbanet du laboratoire CITI de l'INSA de Lyon et s'inscrit dans le cadre du projet ANR-ARESA2. Je tiens donc à remercier tous les membres du laboratoire CITI, mais aussi du département Télécommunications Services et Usage de l'INSA où j'ai enseigné durant ces trois années. Je remercie plus particulièrement tous mes camarades thésards (la liste est trop longue) pour les discussions et bons moments passés ensemble. Je remercie également le personnel administratif pour avoir su me guider dans les méandres des formalités (merci Gaëlle !).

Je remercie ma famille et mes amis pour leur soutien constant tout au long de ces trois années.

Enfin, je remercie Sarah qui a toujours été présente et qui a su me supporter même dans les moments les plus difficiles.

"Never trust a computer you can't throw out a window."
Steve Wozniak

Table des matières

∗. Ce travail est issu des travaux de Master 2 Recherche de Xuan Linh Nguyen que j'ai co-encadrés avec Isabelle Augé-Blum.

Liste des tableaux

Table des figures

Liste d'acronymes

ACP	Algebra of Communicating Processes
AODV	Adhoc On-Demand Vector routing
B-MAC	Berkley - MAC
BER	Bit Error Rate
BPSK	Binary Phase-Shift Keying
CCA	Clear Channel Assesment
CCS	Calculus of Communicating Systems
CDMA	Code Division Multiple Access
CMAC	Convergent MAC
CONTI	CONstant TIme contention resolution
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CSP	Communicating Sequential Processes
CTL	Computation Tree Logic
CT-MAC	Cascading Tournament - MAC
CTS	Clear-To-Send
DAA	Data-Aware Anycast
DAG	Directed Acyclic Graph
DA-MAC	Data Aggregation - MAC
DCF	Distributed coordination function
DMAC	Data-gathering MAC
EDMDP	Event-Driven and Minimum Delay aggregation Path
EECS	Energy Efficient Clustering Scheme
ETSI	European Telecommunications Standards Institue
ExOR	Extremely Opportunistic Routing
FDMA	Frequency Division Multiple Access
FIFO	First In First Out
f-MAC	framelet - MAC
GPSR	Greedy Perimeter Stateless Routing for wireless networks
GRAB	GRAdient Broadcast
GRABUP	GRAdient Broadcast UPdate
I-EDF	Implicit - Earliest Deadline First
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IF	Intermediate Format
LEACH	Low-Energy Adaptive Clustering Hierarchy
LMAC	Lightweight MAC
LOADng	Low power Lossy networks On-demand Ad hoc Distance-vector Routing Protocol - Next Generation
LTL	Linear Temporal Logic
LTS	Labeled Transition System
MAC	Medium Access Control
MANET	Mobile Ad hoc NETwork
MDP	Markov Decision Process

MF-TDMA	Multi-Frequency - Time Division Multiple Access
NTA	Network of Timed Automata
NPTA	Probabilistic Network of Timed Automata
OGDC	Optimal Geographical Density Control
PCCS	Probabilistic Calculus of Communicating Systems
PCTL	Probabilistic Computation Tree Logic
PEDAMACS	Power-Efficient and Delay Aware Medium Access Control protocol
PER	Packet Error Rate
PLTL	Probabilistic Linear Temporal Logic
PPTA	Priced Probabilistic Timed Automata
PR-MAC	Path-oriented Real-time MAC protocol
PTA	Probabilistic Timed Automata
QoS	Quality of Service
R^2A	Real-time and Reliable Agregation protocol
RCsF	Réseau de Capteurs sans Fil
RPAR	Real-time Power Aware Routing
RPL	Routing Protocol for Low-Power and Lossy Networks
RSSI	Received Signal Strength Indication
RTS	Request-To-Send
RTXP	Real-Time X-layer Protocol
RX	Réception
SCV	Système de Coordonnées Virtuelles
SDL	Specification and Description Language
SFEB	Structure-Free and Energy-Balanced aggregation
SINR	Signal-to-Interference and Noise Ratio
SMAC	Sensor MAC
TA	Timed Automata
TCTL	Timed Computation Tree Logic
TDMA	Time Division Multiple Access
TI	Texas Instruments, Inc.
TL-LEACH	Three Layers - Low-Energy Adaptive Clustering Hierarchy
TLTL	Timed Linear Temporal Logic
TLTS	Timed Labeled Transition System
TMAC	Timeout MAC
TRAMA	TRaffic-Adaptive Medium Access protocol
TSA	Timed Safety Automata
TSMP	Time Synchronized Mesh Protocol
TTL	Time To Live
TX	Transmission
UDG	Unit Disk Graph
UML	Unified Modeling Language
WCTT	Worst Case Traversal Time

Chapitre 1

Introduction

1.1 Les réseaux de capteurs sans fil

Les Réseaux de Capteurs sans Fil (RCsF) sont composés de centaines, voire de milliers, de nœuds capteurs autonomes déployés pour mesurer des paramètres physiques de l'environnement. L'architecture d'un nœud est illustrée par la Figure 1.1, ses éléments principaux sont :

- un ou plusieurs capteurs de grandeurs physiques (température, humidité, niveau sonore, etc) ;
- une radio permettant aux nœuds de communiquer ;
- une batterie qui est souvent la seule source d'énergie. Notons que les techniques de récupération d'énergie [Seah et al., 2009] viennent remettre en cause ce paradigme : les capteurs sont capables d'emmagasiner de l'énergie provenant de l'environnement (lumière, vibrations, etc) ;
- un microcontrôleur sur lequel sont reliés les capteurs et la radio. La partie logicielle s'exécute sur le microcontrôleur. Elle est composée d'un système d'exploitation sur lequel s'exécute l'application de mesure et d'une pile protocolaire qui exécute les protocoles réseau.

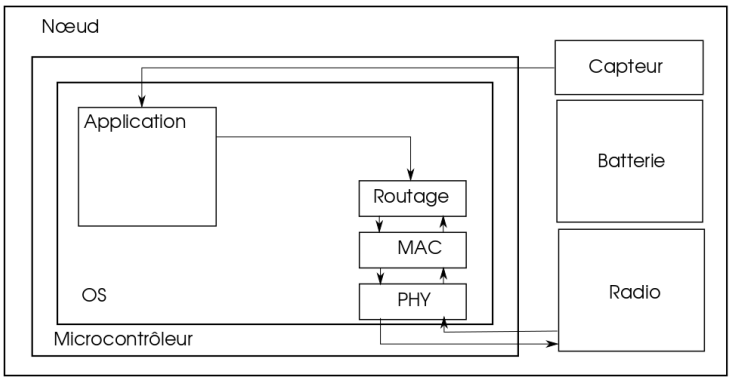


FIGURE 1.1 – Schéma représentant l'architecture générale d'un nœud de RCsF.

Les nœuds sont souvent déployés sur de grandes étendues ou dans des milieux hostiles, cela rend le remplacement des batteries très difficile (sur certains modèles de nœuds la batterie n'est pas remplaçable [Coronis, 2013] car moulée sur la carte). L'économie d'énergie est donc un enjeu majeur, qui est au centre de la recherche dans ce domaine. Les RCsF sont des réseaux radios multi-sauts, généralement sans infrastructure fixe. Du fait du grand nombre de nœuds, il est difficile de les configurer un par un. Ils doivent donc s'auto-organiser pour pouvoir acheminer les informations vers un ou plusieurs points de collecte appelés puits. De même en cas de changement dans la topologie du réseau (nœuds dont la batterie est épuisée ou nœuds détruits, perte de liens radios), les nœuds restant doivent pouvoir se réorganiser de façon autonome.

La réduction de la consommation d'énergie et l'auto-organisation ont donc été les deux thèmes les plus étudiés dans la littérature scientifique. Ces travaux ont fait émerger des mécanismes protocolaires largement adoptés pour acheminer des données dans ce type de réseau :

- au niveau de l'accès au médium, les recherches se sont concentrées sur la réduction de la consommation d'énergie. La technique de cycle d'endormissement [Polastre et al., 2004] qui consiste à éteindre la radio le plus souvent possible pour économiser la batterie s'est imposée ; elle est implémentée dans la plupart des propositions de la littérature [Bachir et al., 2010] ainsi que dans le standard 802.15.4 [IEEE, 2011b] de l'IEEE ;
- au niveau routage, l'auto-organisation du réseau pour l'acheminement des données vers le puits est au centre des problématiques étudiées [Watteyne, 2008]. Dans ce domaine, l'organisation du réseau en gradient [Intanagonwiwat et al., 2000] [Watteyne et al., 2009b] est largement répandue [Al-Karaki and Kamal, 2004] et est à la base de RPL [IETF-ROLL, 2011], protocole de routage standardisé par l'IETF.

Grâce à ces travaux, les RCsF sont maintenant suffisamment matures pour pouvoir aborder de nouvelles problématiques, notamment la qualité de service (noté QoS pour *Quality of Service*). Certaines applications (cf. section 1.2) ont, en effet, des contraintes fortes sur le délai d'acheminement des données et leur taux de livraison. L'acheminement de données sous contraintes applicatives est un domaine qui reste largement ouvert [Chen and Varshney, 2004], notre travail se situe dans ce contexte et consiste d'une part à proposer des mécanismes protocolaires qui permettent le respect des contraintes temporelles et d'autre part à fournir des garanties formelles que ces contraintes sont respectées. Dans la section suivante, nous décrivons les types d'application existants dans le contexte des RCsF ainsi que leurs contraintes applicatives associées.

1.2 Contraintes applicatives

Les applications pour les RCsF sont classées en trois catégories [Tilak et al., 2002] selon la nature du trafic de données quelles génèrent :

- trafic périodique : pour ce type d'applications, les nœuds envoient régulièrement au puits un paquet contenant leur mesure (on peut citer en exemple les applications de relevé de compteurs [Dohler et al., 2007]) ;
- trafic à la demande : les nœuds émettent un paquet sur demande du puits (par exemple, pour les applications de surveillance médicale [Wood et al., 2008], les médecins doivent avoir la possibilité de faire des requêtes pour récupérer les paramètres physiologiques des patients) ;
- trafic événementiel : les nœuds émettent un paquet (appelé alarme dans ce cas) seulement lorsque le paramètre physique observé dépasse un seuil (on peut citer par exemple une application de détection de feux de forêt [Zhang et al., 2007] : au départ de l'incendie, une alarme doit être émise vers le puits).

Les applications citées ci-dessus ont des contraintes différentes. Par exemple, une alarme incendie doit être acheminée par le réseau dans un temps qui est de l'ordre de la minute, alors que pour un relevé de compteur cela peut prendre plusieurs heures. De même, si un relevé de compteur peut être perdu sans conséquence grave, pour une alarme incendie les pertes sont plus problématiques.

En particulier, un rapport technique de l'ETSI [ETSI, 2011] définit des contraintes temporelles cibles pour différentes applications. Le document est centré sur des applications urbaines. Un résumé des applications et de leurs contraintes est donné par le Tableau 1.1.

Application	Périodicité du trafic	Latence du trafic périodique	Latence du trafic apériodique
Relevé de compteur d'eau/gaz	1/jour à 4/heure	300s	30s
Surveillance des conteneurs à ordures	1/heure	NC	NC
Surveillance de la pollution de l'air	1/heure	quelques minutes	1min
Surveillance de l'éclairage publique	1/jour	1min	10s
Système de gestion de parking	1/heure à 4/heure	60s	60s
Système de location de vélos	4/heure à 8/heure	30s	30s

TABLE 1.1 – Exemples de contraintes applicatives tirés de [ETSI, 2011]

Les applications décrites sont de types périodiques et événementiels : par exemple, pour la surveillance de la pollution de l'air, un relevé de chaque capteur par heure permet d'alimenter des cartes de pollution. Quand une pollution chimique dangereuse est détectée un message d'alarme doit atteindre le puits en moins d'une minute. Dans ce dernier cas, il est indispensable que l'alarme atteigne effectivement le puits et respecte l'échéance, les protocoles réseau doivent donc permettre d'acheminer les messages en temps borné et de manière fiable (en évitant les pertes de paquets).

Les applications citées dans [ETSI, 2011] ont de fortes contraintes et certaines (e.g. détection de pollution chimique, détection de fuites de gaz) peuvent être considérées comme critiques. Les applications critiques sont définies comme des applications, dont le mauvais fonctionnement peut avoir des conséquences catastrophiques sur des vies humaines ou sur l'environnement. Dans ce document, nous nous intéressons aux applications critiques temps-réel. Ces dernières, en plus d'avoir un comportement correct (pas de pannes), doivent aussi produire leurs résultats dans un temps borné et connu. Au niveau du réseau, cela impose d'avoir un temps d'acheminement des données borné. Cela pose plusieurs difficultés : d'une part, les protocoles aux niveaux MAC et routage doivent permettre de garantir des bornes temporelles. D'autre part, le lien radio étant non-fiable par nature, il est possible que certains paquets soient perdus. Les protocoles qui supportent des applications critiques doivent limiter les pertes pour répondre aux contraintes des applications visées. De plus, la criticité * des applications impose de prouver formellement leur bon fonctionnement pour se prémunir contre toutes les erreurs possibles. Les techniques de simulation et de test qui n'explorent qu'une partie des comportements du système [Baier and Katoen, 2008] ne sont donc pas suffisantes et doivent être utilisées conjointement avec des méthodes formelles qui permettent de prouver mathématiquement l'absence de mauvais fonctionnement.

Ces problématiques sont très peu abordées dans le contexte des réseaux sans fil [Augé-Blum et al., 2011]. Dans les sections suivantes, nous décrivons les problèmes posés pour l'établissement de communications temps-réel dans les RCsF et pour la vérification formelle de ces protocoles.

1.3 Le temps-réel dans les RCsF

1.3.1 Conception de protocoles temps-réel

On distingue deux types de systèmes temps-réel [Kopetz, 2011] : les systèmes temps-réel strict ou dur (*hard real-time*) et les systèmes temps-réel souple ou mou (*soft real-time*). Pour les systèmes temps-réel dur le non-respect de l'échéance d'un paquet peut avoir des conséquences catastrophiques sur des vies humaines ou sur l'environnement. Par exemple, pour une application de détection de départ de feu de forêt, si l'alarme arrive en retard sur l'échéance, le feu peut alors devenir extrêmement difficile à maîtriser et détruire non-seulement des écosystèmes, mais aussi menacer des constructions et des vies humaines. Pour les applications temps-réel souple, une proportion des messages peut rater l'échéance sans conséquences. C'est le cas pour une application de diffusion de vidéo : lors d'une transmission vidéo des trames contenant des informations sur les images à afficher sont envoyées successivement. Si certaines trames dépassent l'échéance ou sont perdues (on peut tolérer un taux de perte de 0.001% avec la norme MPEG1 [Chen et al., 2004]), la qualité de la vidéo sera dégradée mais l'application pourra continuer à l'afficher. Notre travail porte sur des applications temps-réel dur pour RCsF. En toute rigueur, cela corres-

∗. On définit la criticité comme étant la détermination et hiérarchisation du degré d'importance et de la disponibilité d'un système [Wikipedia, 2013b].

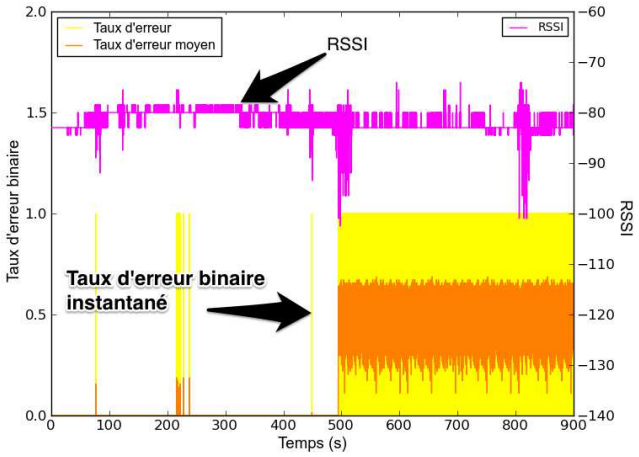


FIGURE 1.2 – Mesure du RSSI et du taux d'erreur binaire entre deux capteurs de la plate-forme SensorLab en fonction du temps

pond à des applications pour lesquelles la borne sur le délai doit être absolument respectée. Le premier problème qui se pose alors est celui de la fiabilité. En effet, les transmissions dans les réseaux sans fil son non-fiables [Goldsmith, 2005], c'est-à-dire qu'il existe une probabilité pour qu'un paquet soit perdu au cours d'une transmission. C'est particulièrement vrai dans les RCsF qui ont des puissances d'émission faibles [Yang, 2010]. La Figure 1.2 illustre les résultats de mesures † de RSSI (*received signal strength indicator* qui correspond à la puissance du signal radio reçu) et de taux d'erreur binaire (nombre de bits perdus sur le nombre de bits transmis) entre un émetteur et un récepteur de la plate-forme SensOrLab conçue dans le cadre du projet ANR ARESA2 [ARESA2, 2013]. Le couple émetteur récepteur est fixe et les nœuds sont espacés d'une cinquantaine de mètres, on constate pourtant que le RSSI et le taux d'erreur binaires ne sont pas stables. A partir de 500 secondes la dégradation du RSSI implique une désynchronisation des nœuds et donc un taux d'erreur instantané égal à 1. Les phénomènes de propagation qui induisent cette variabilité de la puissance de réception et donc les pertes de paquets seront détaillés dans le Chapitre 3. On est alors amené à se poser la question suivante : comment garantir strictement un délai de bout en bout sur un réseau radio non-fiable ? Dans les réseaux filaires, la plupart des analyses de délais de bout en bout [Bauer et al., 2010] [Davis et al., 2007] supposent qu'il n'y a pas d'erreur de transmission car le médium de communication est considéré comme étant fiable. Il est tout de même possible que des erreurs se produisent sur un lien filaire [Ferreira et al., 2004], mais la probabilité de ces événements est suffisamment faible pour les applications visées. De manière générale, des bornes strictes sur le délai dans un système réel sont impossibles à atteindre, car il y a toujours une part de hasard liée aux phénomènes physiques. Il faut donc être capable de comprendre et gérer ce risque. Une borne sur le délai de bout en bout pourra donc

†. Nous remercions Quentin Lampin, ingénieur de recherche à Orange Labs pour nous avoir fourni la figure.

être produite accompagnée d'une probabilité qu'elle soit respectée. Cette probabilité doit être suffisante pour l'application visée [Decotignie, 2012].

Ce constat fait, la deuxième problématique pour garantir des délais d'acheminement dans les RCsF se trouve au niveau protocolaire. En effet, comment concevoir des protocoles qui garantissent le respect de délais de bout en bout (on s'intéresse aux bornes supérieures) et qui assurent la fiabilité des communications tout en prenant en compte les spécificités de ce type de réseau (énergie, large échelle, etc) ? Pour répondre à cette question, il faut d'abord identifier les causes de délais dans ces protocoles. Nous en identifions cinq principales dans la pile protocolaire :

- le temps d'émission ;
- le temps de propagation ;
- le temps d'accès sûr au canal (accès et reprise après collision) ;
- la longueur des routes ;
- les délais d'interaction entre les couches protocolaires.

La durée d'émission d'un paquet est facilement connue : il suffit de connaître le débit de la radio B et la taille du paquet M.

$$D_t = \frac{M}{B} \tag{1.1}$$

S'il existe des paquets de différentes tailles, la durée d'émission est bornée (borne supérieure) par celle du plus grand paquet. Le temps de propagation correspond au temps nécessaire à un bit d'information pour arriver au récepteur à partir de l'instant où il est émis. Cette durée est fonction de la distance émetteur-récepteur et de la vitesse de propagation du signal qui est proche de celle de la lumière. Le temps de propagation est borné par celui de la paire émetteur-récepteur étant les plus éloignés l'un de l'autre. Ces délais sont régis par des équations déterministes et peuvent donc facilement être connus et bornés.

Les fonctions d'une couche MAC sont d'allouer un ensemble de ressources disponibles a un ensemble de nœuds qui ont du trafic à écouler et de résoudre les contentions qui apparaissent lors d'accès concurrents à une ressource. Les ressources peuvent prendre la forme de *slots* de temps, de fréquences, de codes, ou bien de combinaisons de ces types de ressources. Les délais non bornés pour l'accès à une ressource viennent du fait qu'avec certains mécanismes MAC, plusieurs nœuds peuvent entrer en compétition pour la même ressource et tous croire qu'ils l'ont obtenue. Cela produit une collision entre les informations des différents nœuds et ces dernières ne pourront souvent pas être décodées par le ou les récepteurs. Les collisions, dans certains cas peuvent être résolues en temps borné ([Lann and Rivière, 1993] pour les réseaux filaires). Dans d'autres cas, elles sont résolues grâce à des mécanismes qui font appel à des variables aléatoires. Dans ce dernier cas, le délai de résolution n'est pas borné. Pour garantir un délai borné, il faut donc, soit éviter les collisions, soit les résoudre en un temps borné.

Au niveau routage, les mécanismes doivent permettre l'acheminement des données depuis une source quelconque jusqu'au puits. Pour cela des routes sont construites. Une route est une suite de nœuds dont le premier élément est la source du message et

le dernier est le puits. Le délai de bout en bout maximum dans le réseau dépend de la longueur de la plus grande route. En effet, chaque saut a une durée non compressible qui correspond aux durées de transmission, de propagation et d'accès au médium. Si on est capable de borner le nombre de sauts, ainsi que ces durées, on peut donner une borne sur le délai de bout en bout.

Une autre source de délai provient des interactions entre les différentes couches de la pile protocolaire : les délais dans les files d'attente entre les couches. On devra prendre garde à ce que le temps d'attente d'un paquet dans ces files d'attente ait une borne supérieure. Dans ce document, nous nous intéressons surtout aux interactions entre les mécanismes de MAC et de routage, car comme nous venons de le voir, ces deux couches sont déterminantes pour borner le délai d'un saut et la longueur des routes ; et donc le délai de bout en bout. Il faut par conséquent pouvoir garantir qu'après la décision de routage, le paquet obtient un accès au médium en temps borné.

Il est à noter qu'il existe d'autres sources de délai, notamment au niveau logiciel et matériel sur les nœuds. On peut d'abord citer les délais introduits par le système d'exploitation et l'exécution des différentes applications embarquées sur le nœud : le système d'exploitation permet d'ordonnancer les différentes tâches applicatives (opérations sur les données mesurées, décision d'émissions d'alarmes, etc) ainsi que les accès aux ressources matérielles (radio, mémoire de stockage, capteurs, etc). Pour pouvoir garantir le respect d'échéances temporelles, il faut que le système d'exploitation soit temps-réel, c'est-à-dire capable d'ordonnancer les tâches en temps borné. Le système d'exploitation FreeRTOS [FreeRTOS, 2013] permet cela sur des systèmes embarqués à faible mémoire et est utilisé dans le cadre des RCsF [Mazzer and Tourancheau, 2009] [Fraboulet et al., 2007] [Mahmood and Jantti, 2009]. On peut aussi citer le délai de réponse des capteurs, par exemple le capteur de concentration de dioxyde d'azote Alphasense-B4 [Alphasense-B4, 2013] (utilisé pour détecter la pollution au dioxyde d'azote dans les villes) met plusieurs secondes pour détecter une variation de concentration d'un ppm. Dans ce document, nous intéressons seulement aux délais induits par le réseau, et considérons que les autres délais sont connus et bornés.

1.3.2 Vérification formelle de protocoles

Comme mentionné précédemment, les systèmes critiques temps-réel dur doivent fonctionner correctement et respecter des contraintes temporelles. Pour être sûr du respect de ces propriétés, elles devront être vérifiées de manière formelle [Clarke and Wing, 1996]. La vérification formelle d'un système consiste à modéliser le système et ses interactions puis à prouver un ensemble de propriétés sur le modèle. Le modèle d'un système est une représentation mathématique du comportement dudit système. Une propriété est un énoncé qui exprime une spécification voulue pour le système, par exemple dans le cadre d'une application de détection d'événement (comme l'application de détection de feux de forêt précédemment citée) on veut vérifier que "*Le délai de bout en bout d'une alarme incendie ne dépasse pas 10s*". Cette propriété est traduite en langage mathématique (on peut citer les logiques temporelles CTL et LTL [Baier and Katoen, 2008]) pour être vérifiée.

Pour obtenir la vérification formelle d'un système, la preuve de théorème et le *Model Checking* sont les deux approches les plus répandues [Clarke and Wing, 1996]. Pour vérifier des propriétés dans les réseaux, il existe également une autre approche : le *Network calculus* qui est un *framework* théorique pour évaluer les bornes supérieures des délais dans les réseaux. Dans le cas de la preuve, on modélise le système à l'aide d'une algèbre et on se sert des axiomes de cette algèbre pour établir des preuves de propriétés du système. Cela permet de prouver des propriétés très générales, et de travailler sur des systèmes infinis (variables qui prennent des valeurs dans $\mathbb{R}$). Cependant, ce type d'approche oblige souvent à faire des abstractions qui vont éloigner les résultats obtenus de la réalité : on obtient alors des bornes temporelles sur le délai de bout en bout très lâches, c'est-à-dire très éloignées du pire cas réel (mais qui le majorent tout de même). La seconde approche, le *Model Checking*, consiste à représenter le système étudié sous forme d'un système à états et transitions. Un algorithme explore tous les comportements possibles et vérifie que la propriété est vraie dans tout l'espace d'état. Cette approche permet de déceler le cas pire réel. Cependant, si le système étudié est complexe, le nombre d'états à stocker et explorer peut être trop grand pour que l'algorithme donne une réponse en un temps raisonnable. Ce phénomène est appelé explosion (combinatoire) de l'espace d'état.

De nombreux formalismes [Baeten, 2005] permettent de représenter des systèmes et des propriétés. Le formalisme choisi devra avoir un pouvoir d'expression suffisant pour représenter tous les aspects du système (le temps car nous vérifions des propriétés temporelles, les probabilités car le lien est non-fiable, les données pour représenter les échanges de paquets, etc) et les outils qui utilisent ce formalisme doivent avoir un pouvoir d'analyse suffisant pour vérifier les propriétés désirées (incluant le temps, les probabilités).

Des vérifications formelles de systèmes distribuées ont été réalisées avec succès, notamment pour des protocoles pour réseaux filaires [Clarke and Wing, 1996]. Les caractéristiques des RCsF (large échelle, liens non-fiables) rendent l'application de ces méthodes difficile sur ce type de réseaux [Kwiatkowska et al., 2002b] [Fruth, 2006] [Fehnker et al., 2007]. D'abord, les RCsF sont des réseaux sans fil, cela implique que les communications sont de type *broadcast*, c'est-à-dire que seuls les nœuds à portée de l'émetteur reçoivent le paquet (les nœuds de la zone d'interférence le reçoivent, mais ne peuvent pas décoder). Ensuite, la topologie des RCsF est dynamique, les nœuds ne sont pas considérés comme mobiles, mais des nœuds disparaissent lorsque leur batterie s'épuise. La fluctuation des liens radio provoque aussi des changements dans la topologie du réseau. Enfin, les RCsF sont des réseaux larges échelles, ils peuvent être composés de plusieurs centaines de nœuds. Cela pose problème pour le passage à l'échelle des méthodes de vérification.

1.4 Contributions

Nos contributions sont de deux types : d'une part, nous proposons une solution pour l'acheminement de données en temps borné dans les RCsF ; d'autre part, nous

fournissons une méthodologie de vérification formelle des RCsF et l'appliquons à la solution d'acheminement temps-réel proposée.

Notre solution d'acheminement de données temps-réel est composée de trois éléments :

1. un système de coordonnées virtuelles, proposé dans le Chapitre 4, pour organiser le réseau et introduire du déterminisme : ce système permet de déterminer la distance logique (en nombre de sauts) entre un nœud et le puits et de discriminer les nœuds appartenant à un même domaine d'interférence. Ces informations sont utilisées dans les propositions suivantes.

2. un protocole de communications temps-réel pour RCsF, RTXP (Real-Time X-layer Protocol) pour la remontée d'alarmes vers le puits dans le cadre de la détection d'événements dans les RCsF. Il permet de répondre aux problématiques abordées dans la section 1.3.1 :
   - RTXP est *cross-layer* (X-layer), les couches MAC et routage sont conçues ensembles, ce qui permet de gérer les délais d'interactions ;
   - le protocole s'appuie sur le système de coordonnées virtuelles, proposé dans le Chapitre 4, qui permet d'avoir des accès au canal bornés et un choix du nœud relayeur déterministe ;
   - ces coordonnées permettent aussi de connaître le nombre de sauts qui séparent un nœud du puits et ainsi de le borner ;
   - RTXP est localisé : les décisions MAC et routages sont prises sans connaissance de la topologie globale du réseau, cela permet de passer l'échelle ;
   - RTXP s'adapte à la charge de trafic dans le réseau : les nœuds dorment s'il n'y a pas d'alarmes à acheminer, cela permet d'économiser l'énergie ;
   - le protocole utilise un routage opportuniste [Biswas and Morris, 2005] qui permet d'augmenter la fiabilité des communications de bout en bout comme nous le montrons dans le Chapitre 3.

   RTXP est, à notre connaissance, le premier protocole temps-réel MAC et routage localisé pour RCsF, c'est-à-dire qu'il n'est pas nécessaire d'avoir une vision globale de la topologie du réseau pour garantir une borne sur le délai de bout en bout contrairement aux solutions qui allient MAC et routage dans la littérature [Ergen and Varaiya, 2006] [Doherty and Pister, 2008]. De plus, ce protocole tient compte de la consommation d'énergie ce qui n'est pas le cas d'autres propositions temps-réel de la littérature [Caccamo et al., 2002] [Watteyne et al., 2006a] [Roedig et al., 2006].

3. Nous proposons également, dans le Chapitre 6, un protocole d'agrégation qui permet d'atténuer l'effet de tempête d'alarmes qui survient lorsqu'un nombre important de nœuds détectent un phénomène physique et envoient des alarmes aux puits au même instant. Ces tempêtes d'alarmes induisent une surconsommation et une congestion du réseau non nécessaire car l'information qui circule est redondante. Notre proposition permet d'atténuer ce problème en élisant, à l'aide du système de coordonnées virtuelles, un seul nœud pour envoyer l'alarme vers le puits, et ce en temps borné.

Dans un second temps, nous proposons une méthodologie de vérification formelle de protocoles de communication temps-réel pour RCsF. Cette méthodologie est détaillée dans le Chapitre 8 et prend en compte les aspects propres à ces réseaux, comme le lien radio et la taille des RCsF. Nous utilisons l'approche du *Model Checking* qui permet une vérification exhaustive des comportements du système et pour obtenir des bornes sur les délais de bout en bout tout en essayant de limiter l'explosion combinatoire. Dans cette seconde partie, nous proposons les contributions suivantes :

1. Une étude des différentes façons de modéliser la nature *broadcast* du lien radio, qui sont utilisées dans la littérature, montre que certaines augmentent le problème d'explosion combinatoire et doivent donc être évitées.
2. Pour prendre en compte l'aspect probabiliste des communications radio, il faut utiliser un formalisme probabiliste. Les outils qui utilisent ce genre de formalismes ne permettent pas de modéliser les RCsF de manière satisfaisante (restrictions venant du langage de modélisation et des méthodes de vérifications comme détaillé dans le Chapitre 7). Nous développons donc une méthode indépendante de l'outil de vérification pour introduire l'aspect probabiliste du lien radio dans la vérification formelle et pouvoir ainsi estimer le degré de fiabilité du système.
3. Dans la littérature, les vérifications de protocoles de RCsF ne vont pas au delà de quelques nœuds (4 ou 5 en général). Pour passer l'échelle, nous développons une méthode hybride qui mélange l'approche du *Model Checking* et le *Network Calculus*. Le *Model Checking* servant à vérifier localement que chaque nœud à un comportement correct quand il est soumis au comportement global du réseau représenté analytiquement sous forme de courbes de services facilement composables.

La composition de ces techniques constitue une méthodologie de vérification de protocoles temps-réel pour RCsF. Nous l'appliquons pour la vérification de RTXP dans le Chapitre 9.

1.5 Organisation du document

La première partie de ce document est consacrée aux aspects protocolaires temps-réel pour RCsF. Dans le Chapitre 2, nous analysons les protocoles MAC et routage pour RCsF de la littérature, nous nous focalisons en particulier sur les aspects temps-réel de ces protocoles. Comme évoqué dans la section 1.3.1, les transmissions dans les RCsF sont non-fiables. Dans le Chapitre 3, nous étudions donc la fiabilité des transmissions de bout en bout dans les RCsF, nous nous attachons notamment à évaluer la fiabilité des routages classiques et opportunistes pour conclure quant à la méthode la plus adaptée dans le cadre des applications critiques. Dans le Chapitre 4, nous proposons un système de coordonnées virtuelles qui permet d'organiser le réseau et d'introduire du déterminisme dans les mécanismes protocolaires. Le protocole RTXP, présenté dans le chapitre 5, tire parti de ce système de coordonnées virtuelles pour établir des transmissions de bout en bout aux délais bornés dans les RCsF. Dans

le Chapitre 6, nous présentons un mécanisme d'agrégation spatial des alarmes pour limiter le nombre d'informations redondantes acheminées vers le puits par RTXP. Cette proposition utilise également le système de coordonnées virtuelles décrit dans le Chapitre 4.

La deuxième partie de ce document est consacrée à la vérification formelle de propriétés temporelles des protocoles de RCsF, cette partie est complémentaire de la première : elle fournit les outils nécessaires pour garantir formellement que les protocoles proposés offrent bien des délais de bout en bout bornés. Le Chapitre 7 décrit l'état de l'art des méthodes formelles en général et pour les RCsF en particulier : cela nous permet de déterminer les méthodes et outils qui vont être les briques de base de la méthodologie de vérification que nous proposons. Cette méthodologie est décrite dans le Chapitre 8, c'est la contribution principale de cette deuxième partie, elle s'articule en trois points précédemment décrits (cf. section 1.4). Dans le Chapitre 9, cette méthodologie est appliquée à RTXP.

Dans le Chapitre 10 nous concluons sur les travaux présentés dans ce document et donnons des perspectives de travaux futurs.

Ces travaux ont été financés en partie par le projet national ARESA2 ANR-09-VERS-017

Première partie

Protocoles temps-réel pour réseaux de capteurs sans fil

Chapitre 2

Acheminement des données dans les RCsF : vision générale et temps-réel

Dans ce chapitre, nous nous intéressons à l'acheminement des données dans les RCsF : c'est-à-dire les mécanismes qui permettent aux données collectées par les éléments du réseau d'être reçues par le puits. Les deux aspects essentiels pour l'acheminement des données sont le contrôle de l'accès au médium (MAC) et le routage. Le mécanisme MAC doit permettre aux différents nœuds de pouvoir accéder au médium de communication pour écouler leur trafic et de résoudre les collisions lors des accès (concurrence pour une ressource). Le mécanisme de routage crée des chemins depuis les nœuds sources vers le puits. Dans le cadre des RCsF, ces mécanismes doivent prendre en compte les spécificités de ces réseaux, notamment les aspects d'auto-organisation, de consommation d'énergie et de faible capacité du médium [Watteyne, 2008]. Il existe dans la littérature, tant au niveau MAC [Bachir et al., 2010] que routage [Al-Karaki and Kamal, 2004] [Amadou, 2012], un très grand nombre de protocoles qui permettent l'acheminement des données dans ce cadre. Nous présentons dans ce chapitre un état de l'art de ces solutions, nous nous intéressons d'abord aux protocoles qui n'ont pas pour but de garantir une borne temporelle sur le délai d'acheminement des paquets, cela nous permet avoir une vision globale des mécanismes qui répondent au problème de l'acheminement des données dans les RCsF. Nous détaillons les raisons qui font que le délai n'est pas bornable avec ces types de solutions. Puis, nous nous concentrons sur les protocoles qui ont pour but de donner une garantie sur le délai de bout en bout.

2.1 Acheminement de données dans les réseaux de capteurs sans fil

Dans cette section nous décrivons et analysons les techniques de niveaux MAC et routage pour l'acheminement des données dans les RCsF. Ces techniques n'ont pas pour but de réaliser l'acheminement des données en temps borné, mais se concentrent

sur d'autres paramètres (énergie, capacité, etc). Après avoir présenté ces solutions qui rendent possibles les communications dans les RCsF, nous nous intéresserons, dans la section 2.2, aux solutions proposées pour l'acheminement des données en temps-réel sur ce type de réseaux.

2.1.1 Le contrôle d'accès au médium dans les RCsF

Les fonctions d'une couche MAC sont d'allouer un ensemble de ressources disponibles à un ensemble de nœuds qui ont du trafic à écouler et de résoudre les contentions qui apparaissent lors d'accès concurrents à une ressource. Ces mécanismes permettent à chaque nœud d'écouler son trafic. Les ressources peuvent prendre plusieurs formes [Proakis, 2001] : des canaux radios définis par une fréquence porteuse (FDMA), des *slots* de temps (TDMA), des codes (CDMA), ou bien une combinaison des solutions précédentes (MF-TDMA par exemple). Les ressources disponibles sont partagées entre des nœuds, elles leur permettent de communiquer avec leurs voisins. Par contre, si des nœuds interférants utilisent une même ressource, il y a collision et l'information est perdue. Nous commençons donc par définir les termes de nœud interférant, voisinage et 2-voisinage :

Définition 2.1.1 *Un nœud n_a est un* ***nœud interférant*** *pour un nœud n_b si l'activité radio de n_a peut empêcher n_b de recevoir des paquets.*

Définition 2.1.2 *Le* ***voisinage*** *d'un nœud n_a est l'ensemble des nœuds dont n_a peut décoder les communications. On peut noter que des nœuds voisins sont aussi des nœuds interférants.*

Définition 2.1.3 *Le* ***2-voisinage*** *d'un nœud n_a est l'union du voisinage de n_a et des voisinages de ses voisins.*

On note que, dans les RCsF, les interférants, les voisinages et les 2-voisinages évoluent au cours du temps du fait des changements dans la topologie (dus aux disparitions de nœuds et aux liens non-fiables).

Dans le cadre des RCsF, la plupart du temps, on ne dispose que d'un canal radio [Sohraby et al., 2007] pour assurer l'écoulement du trafic, les nœuds qui interfèrent ne doivent donc pas transmettre un paquet au même instant. Par exemple, un problème classique de l'accès au médium pour les réseaux sans fil est celui du terminal caché illustré par la Figure 2.1 : les nœuds A et C veulent envoyer un paquet au nœud B. Les deux émetteurs ne sont pas voisins, s'ils choisissent d'émettre au même instant : le nœud récepteur B ne pourra pas décoder les paquets. Le mécanisme MAC doit donc permettre d'éviter que ces deux nœuds émettent au même instant.

Deux approches sont possibles pour choisir l'instant d'accès au canal : par contention ou réservation préalable. Dans le cas d'un accès par contention, plusieurs nœuds vont entrer en compétition pour la même ressource [IEEE, 2003]. L'autre solution consiste à réserver une ressource pour chaque nœud [Doherty and Pister, 2008]. Cela va influencer le délai d'accès à la ressource et le taux d'utilisation du médium :

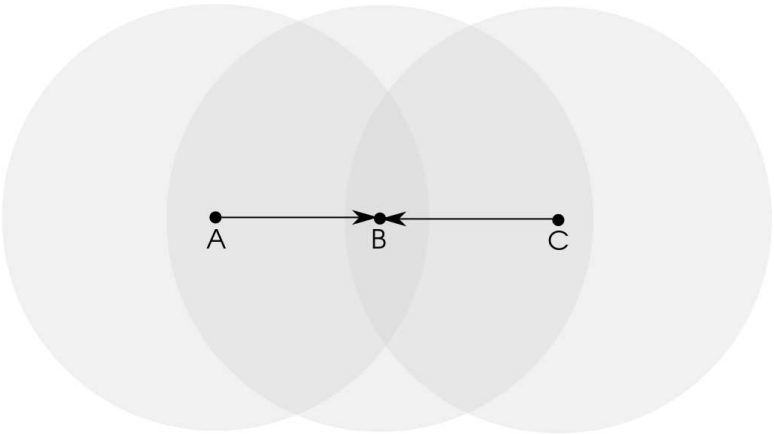
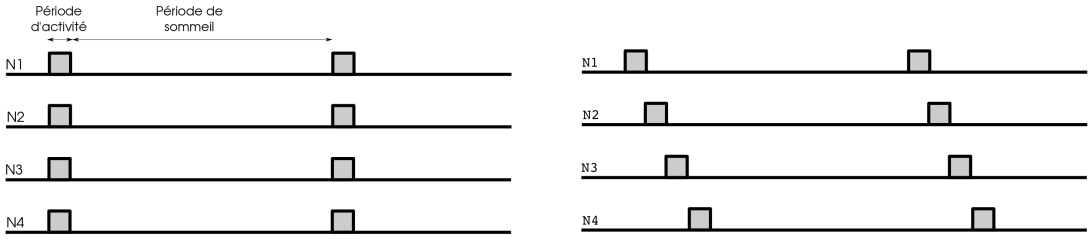


FIGURE 2.1 – Problème du terminal caché : A et C ont un paquet à émettre à B mais ne sont pas à portée radio l'un de l'autre



(a) Eveils simultanés (b) Eveils décalés

FIGURE 2.2 – Exemples d'éveils synchrones : les nœuds partagent la même horloge et connaissent les dates de réveil des autres nœuds.

- les accès par contention permettent un accès plus flexible à la ressource, un nœud qui a besoin d'émettre peut obtenir un accès au canal très rapidement s'il n'y a pas beaucoup de trafic. En revanche s'il y a beaucoup de nœuds qui veulent un accès à la ressource, le délai induit par la compétition retarde l'accès et limite la capacité du médium [Bachir et al., 2010]. L'accès par contention peut être basé sur une compétition dont l'issue est probabiliste [IEEE, 2003] ou bien déterministe [Sobrinho and a.S. Krishnakumar, 1999]. Dans le premier cas, il est possible que la ressource soit allouée à deux nœuds, ce qui cause une collision. Dans le second cas, un seul nœud accédera à la ressource. Ce deuxième cas permet des accès au médium en temps borné [Sobrinho and a.S. Krishnakumar, 1999].
- Dans le cas des accès réservés, un nœud doit attendre la ressource qui lui est dédiée. S'il n'y a pas de trafic, beaucoup de ressources sont inutilisées, mais le nœud doit quand même patienter jusqu'à obtenir sa ressource, cela limite la réactivité du réseau, mais ne met pas en péril le respect des échéances temps-réel.

Des exemples de ces méthodes seront détaillés lors de la description des différents protocoles.

Dans le cadre des RCsF, la couche MAC doit aussi prendre en compte la dynamique du voisinage et l'asymétrie des liens dues au canal radio non-fiable (nous présentons ce point en détails dans la section 3.2.1 du Chapitre 3), ainsi que l'énergie disponible limitée qui réduit la durée de vie du réseau. Pour ce dernier aspect, on constate

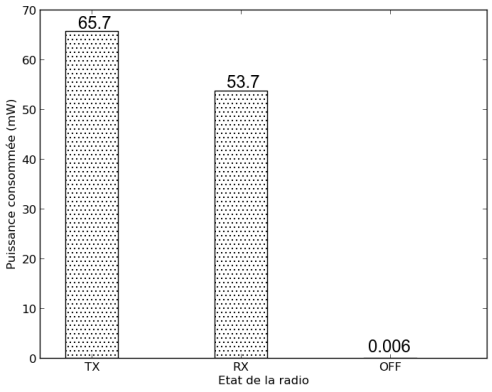


FIGURE 2.3 – Puissance consommée par la radio, données tirées de [Lampin et al., 2012]

que c'est la radio qui consomme le plus d'énergie [Polastre et al., 2004]. La Figure 2.3 illustre ce fait : les consommations d'énergie en mode émission et réception sont 10000 fois plus importantes que lorsque la radio est éteinte (les données sont issues de [Lampin et al., 2012]). Pour réduire la consommation d'énergie, les couches MAC de la littérature utilisent un cycle d'endormissement (*duty cycle*) [Polastre et al., 2004] qui consiste à alternativement allumer et éteindre la radio. La Figure 2.2(a) illustre un cycle d'endormissement : la radio des nœuds est allumée durant la période d'activité et éteinte durant la période de sommeil. La couche MAC doit permettre de gérer ces périodes d'éveils et de sommeil pour que les nœuds voisins soient capables de communiquer. On peut alors classer les protocoles MAC pour RCsF en deux catégories : les protocoles synchrones et ceux asynchrones. Dans le cas des protocoles synchrones, les nœuds connaissent les dates de réveil des autres nœuds (seulement leurs voisins ou tout le réseau). Un mécanisme de synchronisation des nœuds est utilisé : les nœuds partagent une horloge globale ou locale [Elson and Estrin, 2003]. Par exemple, la Figure 2.2(a) représente quatre nœuds qui ont une période d'activité commune et la Figure 2.2(b) quatre nœuds qui ont des périodes d'activité décalées (les protocoles seront détaillés plus loin dans cette section). Dans le cas des protocoles asynchrones, il y a aussi une synchronisation (essentielle pour pouvoir communiquer [Goldsmith, 2005]) mais elle est éphémère et locale : un nœud qui veut communiquer et son voisinage se synchronisent pour le temps de la communication sans échanger les valeurs de leurs horloges. La technique pour réaliser ces synchronisations éphémères et locales est appelée échantillonnage de préambule, elle est décrite par la Figure 2.4 : un préambule est une suite de bits émis avant le paquet de données utiles, sa longueur correspond à une période d'endormissement. Quand un nœud se réveille, il échantillonne le canal (nœuds B et C de la Figure 2.4). Si la présence d'un préambule

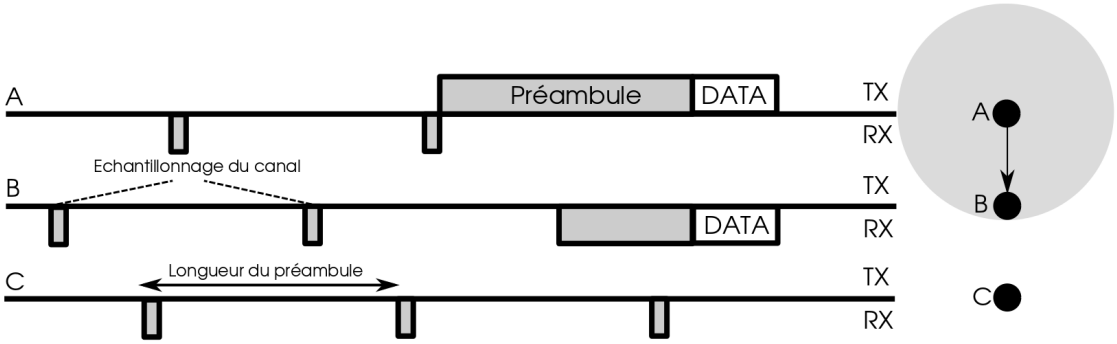


FIGURE 2.4 – Echantillonage de préambule

est détectée, il reste éveillé jusqu'à ce qu'il reçoive le paquet de données (le nœud B reste éveillé pour recevoir le paquet de A). Si aucun préambule n'est détecté, le nœud se rendort pendant une durée égale à la période d'échantillonnage (cas du nœud C). On peut noter que cette technique évite d'avoir à maintenir une synchronisation globale du réseau qui peut paraître coûteuse en énergie. Cependant, il faut émettre un préambule à chaque fois qu'un paquet de données doit être émis (donc à chaque saut d'un paquet qui remonte vers le puits). Une étude réalisée dans [Lampin, 2013] montre qu'en fonction de l'intensité trafic considéré et de la technique de synchronisation globale utilisée, l'échantillonnage de préambule peut s'avérer plus cher en énergie qu'un synchronisation globale.

Les protocoles synchrones

Pour les protocoles synchrones, on distingue différentes méthodes d'accès au médium : soit héritées de 802.11 [IEEE, 2003] ou soit basées sur le mécanisme CONTI [Abichar and Chang, 2005]. On peut noter qu'il existe des protocoles synchrones à accès déterministes [Doherty and Pister, 2008] [Ergen and Varaiya, 2006], ils seront présentés dans la section 2.2 car il permettent des accès au médium en temps borné.

Dans SMAC [Ye et al., 2004], les nœuds se synchronisent de manière distribuée. Un nœud, lorsqu'il se réveille, scrute le canal. S'il reçoit un message de synchronisation (qui contient une date d'éveil), il adopte la date d'éveil correspondante, sinon il émet une date qu'il choisit aléatoirement. Il est possible que plusieurs dates d'éveil se propagent dans le réseau (dans le cas où des nœuds sont démarrés au même instant), mais les auteurs de [Ye et al., 2004] affirment que cette situation est très rare, nous considérons donc que tous les nœuds du réseau ont la même date d'éveil. L'accès au médium se fait par contention, SMAC reprend les mécanismes de 802.11 DCF [IEEE, 2003], notamment, un mécanisme de détection de porteuse (*carrier sense*) et un mécanisme de RTS/CTS pour éviter le problème du terminal caché. Durant le mécanisme de RTS/CTS, un nœud qui veut envoyer un paquet envoie d'abord un RTS (*Request To Send*) qui est une requête d'envoi de petite taille par rapport au paquet de données. Le nœud récepteur, s'il peut recevoir le paquet, répond par un CTS (*Clear To Send*) qui contient l'adresse de l'émetteur. Ce dernier peut en-

suite émettre le paquet de données. L'accès au médium se fait par contention. La technique de CSMA/CA (*Carrier Sense Multiple Access / Collision Avoidance*) est utilisée : un nœud qui veut accéder au canal établit la présence d'un signal sur le canal par mécanisme de un CCA (*Clear Chanel Assessment* : échantillonnage du canal [IEEE, 2003]). Si un signal est détecté, le nœud attend pendant une durée aléatoire (détaillé dans [IEEE, 2003]), si aucun signal n'est détecté le paquet peut être émis. La période d'activité de SMAC est représentée dans la Figure 2.5 : la partie SYNC permet aux nœuds d'échanger des paquets de maintenance de la synchronisation, ensuite deux périodes sont réservées pour émettre les messages RTS et CTS, enfin une période est réservée pour l'envoi du paquet de données. Le principal inconvénient de ce type de solutions est que les nœuds doivent se réveiller même s'il n'y a pas de trafic dans le réseau. Cependant, cela permet de réaliser des communications *broadcast* car tous les voisins d'un nœud sont prêts à recevoir un paquet au même instant. TMAC [van Dam and Langendoen, 2003] reprend le concept général de SMAC mais le cycle d'endormissement est adapté au trafic à écouler : un nœud reste éveillé le temps nécessaire pour écouler son trafic et se rendort s'il ne reçoit pas de trafic durant une durée prédéterminée (cf. [van Dam and Langendoen, 2003] pour la définition et paramétrisation de cette durée). Les auteurs montrent que cette approche permet de réduire la consommation d'énergie de SMAC d'un facteur 5. Dans DMAC [Lu et al., 2004], les auteurs proposent que les dates de réveil des nœuds soient fonction de la distance au puits. Cela permet de réduire le délai de convergence des paquets vers le puits. Ce mécanisme est décrit par la Figure 2.2(b) : $N4$ est le nœud le plus proche du puits, il se réveille donc en dernier. Un paquet provenant de $N1$ peut atteindre $N4$ en moins d'une période d'endormissement. Avec SMAC, pour faire plusieurs sauts un paquet est retardé de plusieurs périodes de sommeil.

IEEE 802.15.4 [IEEE, 2011b] est un standard qui définit une couche MAC pour RCsF. Les nœuds peuvent s'associer entre eux pour former des topologies maillées ou en étoiles. Dans les deux cas, au moins un nœud est coordinateur et synchronise les autres grâce à des *beacons* (balises émises à intervalles réguliers). Les nœuds se réveillent à chaque *beacon*, exécutent une période d'activité et se rendorment jusqu'au prochain *beacon*. La période d'activité de 802.15.4 est représentée dans le Figure 2.5 : elle est composée, après le *beacon*, d'une période de contention (*CAP : Contention Access Period*) et d'une période sans contention (*CFP : Contention Free Period*). Durant la période CAP les nœuds qui ont des paquets à émettre accèdent aux *slots* en CSMA/CA. Durant la CFP les nœuds disposent de *slots* garantis où ils sont sûrs de pouvoir émettre. Cette période, contrairement à la CAP, permet de garantir le respect de bornes temporelles, elle sera donc abordée dans la section 2.2.1.

CT-MAC [Lampin, 2013] est un protocole synchrone pour RCsF basé sur le mécanisme de CONTI [Abichar and Chang, 2005]. Les nœuds qui ont du trafic à écouler participent à une compétition pour l'accès aux ressources. Le protocole opère en quatre phases, son organisation est illustrée par la Figure 2.6 (dans le cas où trois ressources sont disponibles) : il y a deux phases de compétition, une phase d'annonce pour que les nœuds qui ont obtenu des ressources indiquent qu'ils vont transmettre et une phase d'utilisation des ressources. Lors de la première phase, chaque nœud choisit aléatoirement un *slot* dans une fenêtre de contention de taille CW_1 divisée en

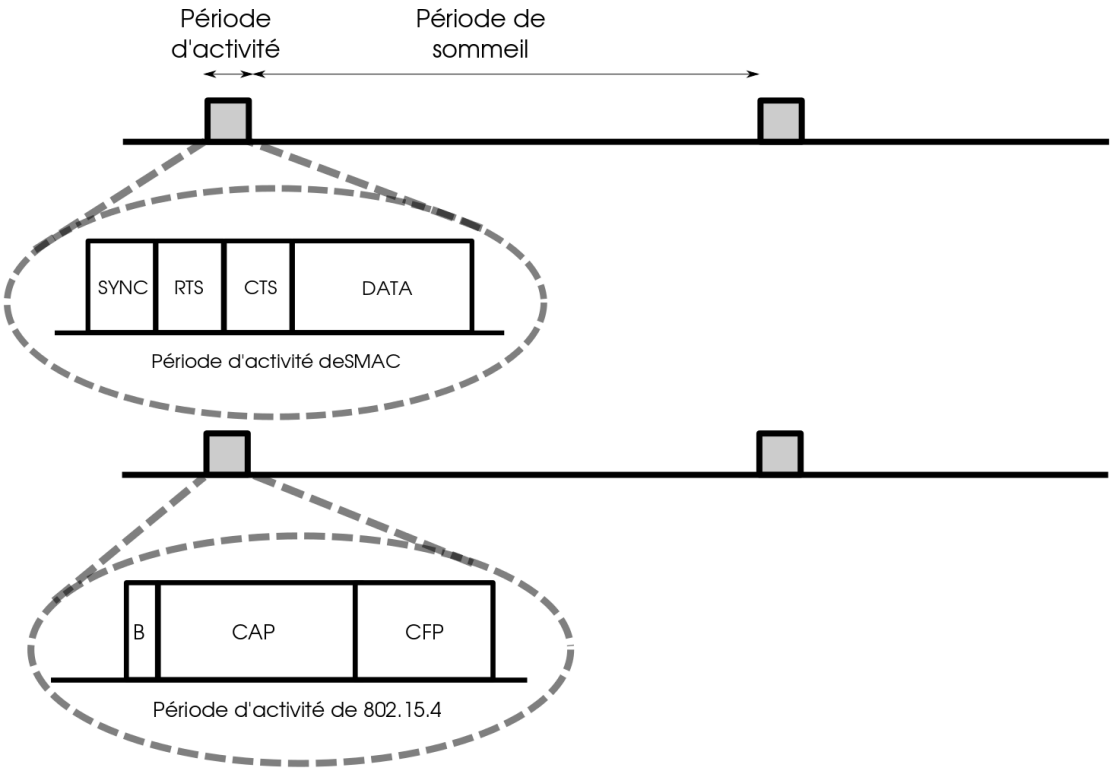


FIGURE 2.5 – Exemples de périodes d'activités : SMAC et IEEE 802.15.4

K_1 *slots* de temps. Il émet un signal d'occupation (suite de bits aléatoire) dans le *slot* choisi si $C_{rank} \geq C_{log}$, C_{rank} étant le nombre de *slots* occupés dans la fenêtre de contention avant le *slot* choisi et C_{log} étant le nombre de ressources disponibles. Dans ce cas, il participe à une deuxième compétition pour la ressource numéro C_{rank} (si $C_{rank} \geq C_{log}$ le nœud se rendort). La deuxième partie de la compétition consiste en un mécanisme de CONTI qui est effectuée pour chaque ressource : une fenêtre de taille CW_2 est divisée en K_2 *slots* de temps, pour chaque *slot* un nœud a une probabilité p d'émettre un signal et $1 - p$ de faire un CCA, si un signal est détecté le nœud se retire de la compétition. Un nœud qui ne se retire pas de la compétition jusqu'à la fin de CW_2, gagne l'accès à la ressource. L'utilisation du mécanisme de CONTI permet de meilleures performances que 802.11 DCF, notamment, un meilleur débit et un taux de collision plus faible [Abichar and Chang, 2005]. La compétition en deux étapes utilisée par CT-MAC permet d'améliorer ces paramètres [Lampin, 2013]. Par ailleurs, un atout majeur de CT-MAC est sa capacité à allouer plusieurs ressources en une seule compétition (la durée de la compétition dépend tout de même du nombre de ressources à allouer).

On note, pour conclure sur les protocoles synchrones, que les protocoles décrits dans cette section ne permettent pas de garantir un accès à la ressource en temps borné (excepté les CFP de 802.15.4 qui seront abordé dans 2.2.1). En effet, même si les mécanismes de CT-MAC ont de très bonnes performances, il y a une probabilité

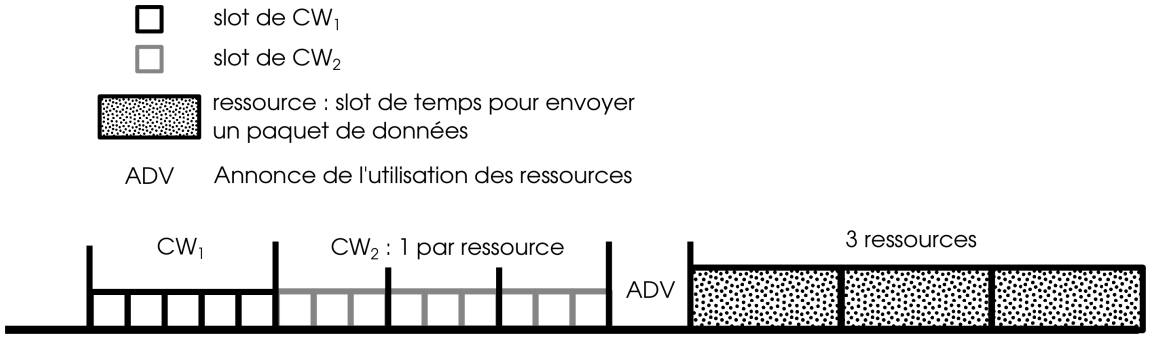


FIGURE 2.6 – Vue d'ensemble de l'organisation temporelle de CT-MAC dans le cas où trois ressources sont disponibles

non nulle qu'une ressource soit attribuée à plusieurs nœuds. On ne peut donc pas borner strictement le temps d'accès au médium avec ce type de protocoles.

Les protocoles asynchrones

Pour les protocoles asynchrones, les dates de réveil des nœuds sont distribuées aléatoirement. Pour échanger des paquets, les nœuds doivent être éveillés au même instant, donc être temporairement synchronisés. Pour réaliser ces synchronisations temporaires, les protocoles de la littérature utilisent la technique d'échantillonnage de préambule décrite précédemment et illustrée par la Figure 2.4. Cette technique est à la base des protocoles présentés dans cette section.

B-MAC [Polastre et al., 2004] est une couche MAC qui gère les accès par contention avec la technique de CSMA/CA. B-MAC se focalise sur la qualité du CCA, cela permet d'éviter que les nœuds ne restent éveillés quand aucun préambule n'est émis. Une technique de contrôle de gain est utilisée pour calculer avec précision le niveau du bruit de fond. Les nœuds échantillonnent le canal quand il est censé être inoccupé. Les échantillons sont ensuite filtrés pour estimer le niveau du bruit de fond. B-MAC propose aussi la possibilité de changer dynamiquement l'intervalle d'échantillonnage en fonction du trafic dans le réseau. Avec B-MAC, les nœuds reçoivent en moyenne la moitié d'un préambule avant de recevoir le paquet de données utiles. Pour réduire la consommation d'énergie due au préambule, des améliorations de la méthode d'échantillonnage de préambule ont été proposées.

WiseMAC [Enz et al., 2004] permet de réduire la taille du préambule pour les communications *unicast*. Un nœud peut mémoriser la date de réveil de ses voisins. Lorsqu'il doit envoyer un paquet à l'un de ses voisins, il émet un préambule à l'instant qui correspond au réveil du voisin en question. La longueur du préambule dépend de la qualité des horloges des nœuds. En effet, si les horloges divergent peu, le préambule peut être très court. Il doit être assez grand pour absorber le maximum de la dérive entre les horloges [Elson and Estrin, 2003]. Cette technique réduit la durée d'écoute du préambule, mais si deux nœuds ont des dates de réveil proches, ils vont rester éveillés pour les deux paquets et donc en partie pour du trafic qui ne leur est pas adressé.

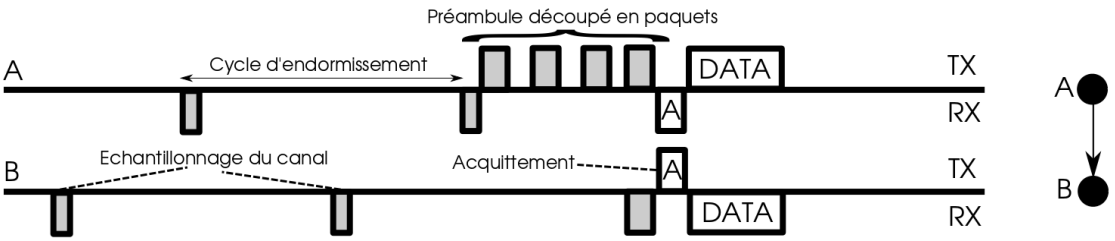


FIGURE 2.7 – Exemple illustrant le fonctionnement de X-MAC : le nœud A transmet un paquet au nœud B

X-MAC [Buettner et al., 2006] revoit le concept de préambule en introduisant la notion de préambule "paquétisé". Le préambule long est découpé en une alternance de paquets et de périodes de contention, les paquets contiennent l'adresse du destinataire, et les périodes de contention permettent d'envoyer un acquittement aux paquets de préambule. Quand un nœud doit envoyer un paquet, il émet des paquets de préambule contenant l'adresse de destination. Le nœud destination, quand il se réveille, répond avec un acquittement au paquet de préambule, cela stoppe l'émission du préambule et le paquet de données utiles est émis. La Figure 2.7 illustre un exemple d'exécution de X-MAC, A étant le nœud source et B le nœud destination. On constate que le délai pour transmettre le paquet est inférieur à la durée d'un cycle d'endormissement plus la durée d'émission du paquet (alors qu'avec BMAC, il faut toujours attendre la fin du préambule pour transmettre).

ContikiMAC [Dunkels, 2011] reprend les mécanismes introduits par B-MAC, WiseMAC et X-MAC. En effet, ContikiMAC utilise un préambule découpé en paquets. Cependant, au lieu d'envoyer des paquets courts de préambule comme X-MAC, l'émetteur envoie directement les données durant le préambule. Quand un nœud se réveille, il a le même comportement que dans X-MAC, sauf qu'il acquitte directement le paquet de données utiles. Un mécanisme de mémorisation des dates de réveil des voisins similaire à WiseMAC est implémenté. Un mécanisme de ré-endormissement rapide est utilisé pour limiter l'impact des faux-positifs du CCA. Les nœuds se rendorment s'ils détectent un signal pendant une durée qui ne correspond à aucun schéma de communication du protocole. Le signal détecté est alors considéré comme du bruit et le nœud se rendort. Tous ces mécanismes permettent de réduire la consommation d'énergie. Cependant, avec ce type de protocoles, le *broadcast* coûte cher en énergie car il faut synchroniser tous les voisins de l'émetteur avant de transmettre (ou bien transmettre plusieurs fois).

Comme les protocoles synchrones présentés précédemment, les protocoles présentés dans cette section utilisent un accès aléatoire au médium. Ils ne permettent donc pas de garantir une borne stricte sur le temps d'accès. De plus, ces protocoles se focalisent plus sur l'amélioration du mécanisme de préambule plutôt que sur la méthode d'accès au médium en elle-même qui est peu détaillée. En effet, dans la plupart des cas l'accès est aléatoire, mais les articles présentés ne détaillent pas la gestion des collisions.

2.1.2 Le routage dans les RCsF

Le but d'un protocole de routage est de construire (et éventuellement maintenir) des routes pour acheminer des paquets entre des sources et des destinations. Dans le cadre des RCsF, nous nous intéressons principalement au trafic qui a pour source n'importe quel nœud du réseau et pour destination le puits : ce type de trafic est nommé *convergecast*. Nous proposons de classer les protocoles de routage pour RCsF en quatre catégories : les protocoles hiérarchiques [IETF-ROLL, 2011], les protocoles à la demande [Perkins and Royer, 1999], les protocoles aléatoires [Servetto and Barrenechea, 2002] et les protocoles *greedy* [Karp and Kung, 2000].

Routage hiérarchique

Les protocoles de routages hiérarchiques organisent le réseau en une topologie logique : des nœuds parmi les voisins physiques (nœuds à portée radio) sont choisis comme voisins logiques dans la structure de routage (topologie logique). Les structures de routage regroupent les nœuds sous forme de *clusters* ou bien les organisent en arbres. Dans le premier cas, un nœud du *cluster* appelé *cluster-head* collecte les données de tous les membres du *cluster*, puis il transmet ces données au puits.

LEACH [Heinzelman et al., 2000] est un protocole qui organise le réseau en *clusters*. Les *cluster-heads* agrègent le trafic de tous les nœuds appartenant au *cluster*, ils consomment donc beaucoup plus d'énergie que les autres nœuds. LEACH propose de répartir la consommation d'énergie en sélectionnant de façon aléatoire les *cluster-heads* pour une période de temps prédéterminée (nommée cycle) : au début de chaque cycle, les nœuds déterminent aléatoirement et localement s'ils sont *cluster-head* ou non. Ils déclarent leur statut grâce à un message envoyé en *broadcast*. Les nœuds non *cluster-heads* choisissent de se rattacher au *cluster-head* qui demande le moins de consommation énergie pour être joint. Chaque *cluster-head* produit ensuite un ordonnancement pour les communications des nœuds appartenant à son *cluster*. Il reçoit ensuite les paquets des membres du *cluster*, les agrège et les transmet directement au puits. Le fait que tous les *clusters* doivent pouvoir atteindre le puits en un saut rend difficilement implémentable ce genre de protocoles pour les réseaux très larges échelles. De plus, le changement de *cluster-head*, même s'il permet de répartir la consommation d'énergie entre les nœuds, a un coût en trafic de signalisation qui se répercute en coût énergétique. Des améliorations ont été proposées, notamment TL-LEACH [Loscri et al., 2005] qui propose une organisation en deux niveaux de clusters. Les *cluster-head* secondaires transmettent leur trafic aux *cluster-head* primaires qui transmettent au puits. Cela réduit le nombre de nœuds qui doivent directement transmettre au puits et améliore donc le passage à l'échelle au prix d'une organisation plus complexe et donc plus coûteuse en énergie. Le protocole EECS [Ye et al., 2005b], reprend l'organisation en *clusters* de LEACH, mais les *clusters* sont formés en fonction de l'énergie résiduelle de chaque nœud pour augmenter la durée de vie du réseau.

Dans le cas d'une organisation en arbre, le puits est la racine. Chaque nœud choisit un parent, qui est plus proche que lui du puits, et lui transmet son trafic. Le choix du parent dépend de la métrique de routage utilisée.

RPL [IETF-ROLL, 2011] est un protocole de routage pour les réseaux IPv6 basse consommation et non-fiables, standardisé par l'IETF. RPL utilise les graphes orientés acycliques (noté DAG pour *Directed Acyclic Graphs*) pour représenter la topologie de routage. Un DAG est enraciné à un point de collecte (puits). Il est construit de proche en proche : le puits annonce qu'il est la racine du DAG, ses voisins mettent à jour leurs tables de routage et diffusent l'information. Un nœud choisit ses parents dans le DAG en fonction de la métrique à optimiser (latence, fiabilité, énergie, etc.). Plusieurs instances de DAG optimisant plusieurs métriques peuvent coexister. Les paquets seront routés selon les besoins de l'application.

Les protocoles basés sur les *clusters* et les arbres permettent de maîtriser le nombre de sauts pour atteindre le puits et donc de maîtriser les délais de bout en bout. Cependant l'énergie dépensée pour la maintenance de ces structures est très importante dans les réseaux large échelle [Amadou, 2012]. De plus, l'interfaçage avec les couches MAC peut poser problème car il faut pouvoir faire correspondre les structures topologiques de niveau MAC et de niveau routage (notamment, le cas de RPL et 802.15.4 est évoqué dans la section 2.1.3).

Routage à la demande

Le routage à la demande consiste à créer une route depuis une source vers une destination, par un mécanisme de requête, lorsque cela est nécessaire. AODV [Perkins and Royer, 1999] est l'exemple classique de protocole à la demande. Les routes ne sont maintenues que si elles sont actives, c'est-à-dire qu'il y a du trafic échangé. Quand un nœud veut communiquer avec une nouvelle destination, il inonde le réseau avec un paquet RREQ (route request) et attend un message RREP (route reply). La destination ou bien un nœud qui a une route active vers la destination peut répondre au message RREQ par un RREP. Ce type de protocole est très coûteux en énergie du fait du mécanisme d'inondation. Cependant, il permet de pouvoir router des informations dans des environnements dynamiques, car une route qui devient obsolète du fait d'un changement dans l'environnement sera reconstruite. Ce type de protocole est donc plus adapté aux réseaux MANET qui sont sujets à des changements de topologie importants dus à la mobilité, et qui sont moins contraints en énergie.

Le protocole LOADng [IETF, 2013] en cours de standardisation à l'IETF étend AODV tout en allégeant certains mécanismes pour réduire la consommation d'énergie. Ce type de routage reste peu approprié aux RCsF [Lu et al., 2008] sauf dans le cas où l'environnement radio est très dynamique et le trafic est faible [IETF, 2013].

Ce type de routage peut permettre de respecter des bornes temporelles strictes si la longueur de la route produite la plus longue est connue. Il faut aussi que la construction de la route puisse se faire en temps borné, ce qui n'est pas garanti par AODV ou LOADng.

Routage aléatoire

Le routage aléatoire consiste à choisir le prochain saut d'un paquet de manière aléatoire dans la liste des voisins disponibles. Le paquet est réémis jusqu'à ce qu'il atteigne le puits (ou que le champ TTL - *Time To Live* - qui est décrémenté à chaque saut, atteigne la valeur 0).

Dans [Servetto and Barrenechea, 2002], les auteurs définissent un protocole de marche aléatoire. Le but est de router des paquets sur un réseau maillé large échelle en ayant le plus de partage de charge possible. L'environnement est dynamique, les nœuds s'endorment et se réveillent à des instants aléatoires. Les auteurs montrent que leur solution permet un partage de charge statistique au niveau global du réseau, en se basant uniquement sur des informations et des décisions locales.

Ce type de routage est résilient [Erdene-Ochir et al., 2010], c'est-à-dire qu'il permet de s'adapter aux changements brusques et importants de la topologie du réseau. Cependant, la longueur des routes n'est pas du tout maîtrisée. Des phénomènes d'oscillations peuvent apparaître (paquet qui est routé vers le nœud duquel il vient indéfiniment), cela augmente énormément le temps moyen pour atteindre la destination. Pour palier aux oscillations, des protocoles de marches aléatoires avec mémoire ont été proposés [Altisen et al., 2012]. Le paquet contient une liste de nœuds qui ne peuvent pas être choisis comme prochain saut, ce sont généralement les nœuds qui viennent d'être visités. Les auteurs de [Altisen et al., 2012] prouvent que le temps moyen pour atteindre la destination (le nombre de sauts en fait) est fini. Cette information n'est cependant pas suffisante pour garantir des délais de bout en bout, le pire cas étant infini.

Routage *greedy*

Les algorithmes de routage *greedy* prennent des décisions de routages qui sont les meilleures localement. C'est-à-dire que le meilleur voisin est choisi comme prochain saut vers le puits. Pour évaluer les différents choix possibles, une métrique locale de routage est utilisée. Une métrique classique et largement utilisée est la distance du prochain saut par rapport au puits, le nœud qui est le plus proche géographiquement du puits est choisi comme prochain saut : on parle alors, dans ce cas, de routage géographique.

GPSR [Karp and Kung, 2000] est un protocole de routage géographique. Les nœuds connaissent leur position GPS et l'envoient régulièrement, en *broadcast* à leurs voisins. Chaque nœud maintient une liste de ses voisins et de leurs coordonnées. Quand un nœud a un paquet à envoyer ou relayer vers le puits, il choisit le voisin qui permet d'avancer le plus possible dans la direction du puits. Cet algorithme a cependant un défaut, il est possible de tomber dans un minimum local : c'est-à-dire un nœud qui n'a pas de voisin géographiquement plus proche du puits que lui comme représenté sur la Figure 2.8. Ce problème est dû à l'existence de trous dans le réseau [Karp and Kung, 2000]. Dans ce cas, un autre algorithme est utilisé : le *face routing*. Les paquets sont routés sur une des faces du trou comme représenté sur la Figure 2.8. Le prochain saut est alors choisi en utilisant la règle de la main gauche (ou droite), le

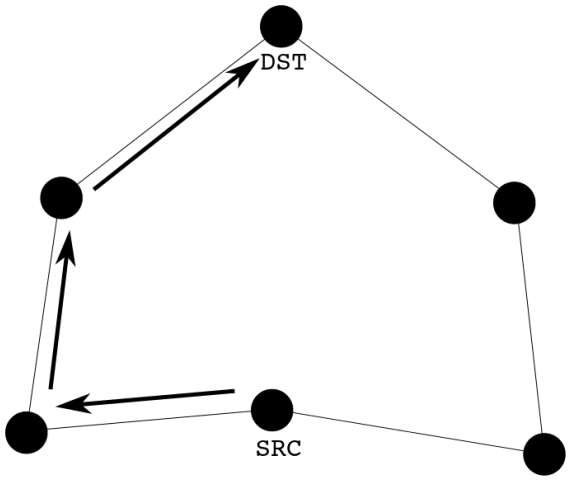


FIGURE 2.8 – Comportement de face routing

paquet est envoyé au premier voisin qui rencontre l'axe source-destination subissant une rotation centrée sur la source dans le sens inverse des aiguilles d'une montre (pour la règle de la main gauche). Ce type de protocole a l'avantage de nécessiter très peu d'informations sur la topologie globale du réseau. En effet, il suffit que les nœuds connaissent la position du puits, leur position ainsi que celles de leurs voisins. Cependant, le coût financier et la consommation énergétique des modules GPS est un frein au développement de ce type de routage. De plus, [Watteyne et al., 2007] montre qu'une imprécision dans la position des nœuds peut mener à des erreurs, lors de l'exécution de l'algorithme de *face routing*, qui empêchent la livraison des données.

Pour palier à ces problèmes, des systèmes de coordonnées virtuelles ont été proposés. Ces systèmes sont basés sur les relations de voisinage radio plutôt que sur les positions géographiques réelles. Un nœud obtient sa position logique dans le réseau en échangeant des messages avec ses voisins. Dans [Rao et al., 2003], les auteurs proposent un système de coordonnées cartésien. Les coordonnées sont calculées dans un espace à deux dimensions. Dans une première version, la position d'une partie des nœuds est connue. Ces nœuds sont sur le périmètre du réseau. Au départ, tous les nœuds (non nœuds périmètres) ont leur coordonnée initialisée à $(0, 0)$. A chaque itération de l'algorithme proposé, les nœuds voisins échangent leurs coordonnées. Ils calculent ensuite leurs nouvelles coordonnées comme étant le barycentre des coordonnées de leurs voisins. Les coordonnées vont donc se disperser sur l'espace entre $(0, 0)$ et le périmètre où sont placés les nœuds dont la position est connue. Dans une deuxième version, les positions des nœuds périmètres ne sont pas connues. Elles sont données arbitrairement en mappant des nœuds sur un cercle virtuel qui contient le réseau. L'algorithme de mise à jour itératif est ensuite exécuté. Les auteurs de [Watteyne et al., 2009a] améliorent le mécanisme d'obtention des coordonnées de [Rao et al., 2003] en utilisant le trafic de données utiles pour initialiser et mettre à jour les coordonnées. Dans [Cao and Abdelzaher, 2004], des coordonnées basées sur le nombre de sauts sont présentées. Les nœuds prennent pour coordonnées le nombre

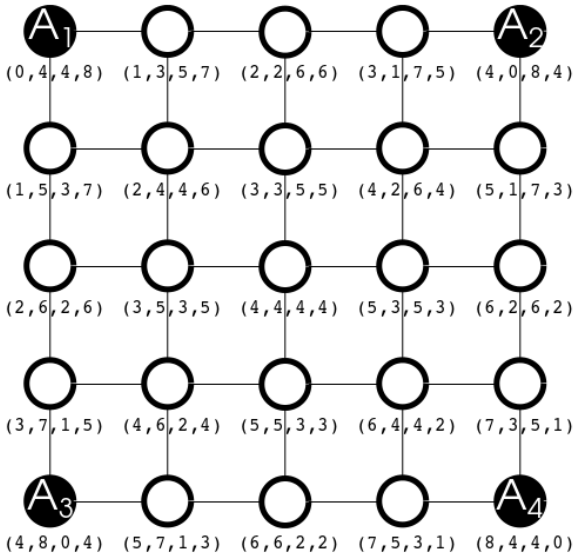


FIGURE 2.9 – Système de coordonnées avec ancres, avec (i, j, k, l) respectivement les distances aux ancres A_1, A_2, A_3 et A_4

de sauts qui les séparent de nœuds ancres. Les ancres inondent le réseau avec un message d'initialisation contenant un compteur initialisé à 0. Chaque nœud relaye ce message et incrémente le compteur. La valeur du compteur la plus petite est conservée comme coordonnée. Le nombre d'ancres détermine la dimension de l'espace des coordonnées. La Figure 2.9 montre un exemple de système de coordonnées avec quatre ancres (nœuds noirs) dans un réseau grille. Les nœuds sont donc placés dans un espace à quatre dimensions (d'entiers naturels). Un routage de type *greedy* pourra ensuite utiliser ces coordonnées.

GRAB [Ye et al., 2005a] est un protocole de routage par gradient qui tire parti de la nature *broadcast* des transmissions radios. À l'initialisation du réseau, chaque nœud récupère le nombre de sauts qui le sépare du puits par un mécanisme d'inondation (il est d'usage d'appeler cette valeur "gradient" du nœud, nous utilisons cette appellation dans la suite de ce document). C'est un cas particulier des coordonnées basées sur les ancres avec une seule ancre : le puits. Un routage de type *greedy* opportuniste est ensuite utilisé : un nœud qui veut envoyer un paquet au puits, transmet ce paquet en *broadcast* à ces voisins, un des voisins qui reçoit le paquet est ensuite élu pour le retransmettre. Cette élection est basée sur le gradient : un nœud qui est plus proche du puits (en nombre de sauts) que l'émetteur pourra relayer le paquet. Plusieurs nœuds peuvent avoir le même gradient, une métrique est donc utilisée pour choisir le relayeur (force du signal, niveau d'énergie, aléatoire, ...). Un ou plusieurs relayeurs peuvent être sélectionnés en fonction de la fiabilité désirée. On peut noter que le fonctionnement en *broadcast* nommé routage opportuniste (décrit en détails dans la section suivante) permet d'augmenter la fiabilité car il faut que seulement un nœud sur l'ensemble des voisins reçoive le paquet pour qu'il ne soit pas perdu. La Figure 2.10 représente un exemple de fonctionnement de GRAB, seuls les nœuds plus proches du

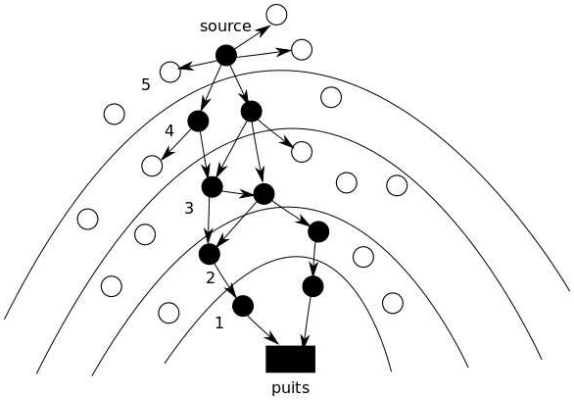


FIGURE 2.10 – Fonctionnement de GRAB

puits relayent le paquet. Il est dupliqué une fois, puis les deux instances sont routées vers le puits (les chiffres représentent le gradient).

Le routage *greedy* est intéressant pour les RCsF car il ne nécessite pas de connaissance globale du réseau et ne se base que sur des informations sur le voisinage. Cependant, pour le cas du routage géographique, il est difficile de borner la longueur d'une route. En effet, les trous dans le réseau vont rallonger la route. En revanche, les approches avec des coordonnées virtuelles basées sur les ancres, permettent de connaître à l'avance le nombre de sauts qu'un paquet va effectuer depuis n'importe quelle source et donc de contraindre la longueur des chemins. Ce point sera abordé dans la section 2.2. Avant d'aborder les solutions *cross-layer* qui définissent à la fois des mécanismes aux niveaux MAC et routage, nous regardons les protocoles de routage sous un autre angle : nous distinguons les protocoles de routage dits classiques des protocoles de routage opportunistes.

Routage classique et routage opportuniste

La nature *broadcast* des transmissions radios peut être vue comme un inconvénient : elle génère des collisions et des paquets adressés à un nœud en particulier sont reçus par d'autres nœuds qui consomment donc inutilement de l'énergie. Cependant, comme nous l'avons vu dans la description de GRAB, on peut tirer parti de cet aspect pour augmenter la fiabilité du routage, c'est le but des protocoles opportunistes [Biswas and Morris, 2005] [Lampin et al., 2012] [Ye et al., 2005a]. Avec les protocoles de routage opportunistes, le prochain saut est choisi après l'émission du paquet par un ensemble de relayeurs potentiels. Cela crée une diversité spatiale : des routes alternatives peuvent être utilisées, et donc cela augmente la probabilité de succès du saut. Dans le cas du routage classique, où le prochain saut est choisi avant l'émission du paquet, il n'y a qu'une seule alternative : si le nœud choisi ne reçoit pas le paquet, le saut échoue.

Les auteurs [Biswas and Morris, 2005] introduisent la notion de routage opportuniste avec le protocole ExOR : un nœud émetteur indique dans le paquet, une liste de relayeurs potentiels classés par ordre de priorité pour relayer le paquet et le transmet en *broadcast*. Sur réception du paquet, un nœud vérifie s'il est dans la liste. Si c'est le

cas il déclenche une temporisation dont la durée dépend de la place du nœud dans la liste (plus il est loin dans la liste, plus la temporisation est longue). Après écoulement de la temporisation, le nœud émet le paquet s'il n'a pas déjà été émis. Si le nœud n'est pas dans la liste, il ne participe pas au relayage du paquet. Les auteurs montrent que ce type de routage permet de réduire le nombre de retransmissions, car il augmente la fiabilité des transmissions. En revanche, ce protocole souffre du problème de réplication de paquets : plusieurs relayeurs potentiels relayent le paquet, cela induit une consommation d'énergie inutile.

Dans [Lampin et al., 2012], les auteurs adaptent la notion de routage opportuniste au routage sur une structure DAG. Comme pour RPL [IETF-ROLL, 2011], un DAG enraciné au puits est créé et maintenu par les nœuds, chaque nœud reçoit de son parent dans le DAG une adresse et un domaine d'adresses qu'il peut partager entre ses fils. Les nœuds émettent leurs paquets en *broadcast*. Sur réception d'un paquet, un nœud vérifie si l'adresse de l'émetteur est dans son sous-domaine (si le nœud est un ancêtre de l'émetteur dans le DAG). Si c'est le cas, il participe à une phase de contention découpée en *slots* de temps : le nœud choisi un *slot* en fonction de la taille de son sous-domaine : plus il est grand, moins il attend (plus le sous-domaine d'un nœud est grand, plus il est proche du puits dans le DAG). Les auteurs montrent que leur solution a de meilleures performances que RPL en termes d'énergie et de délai, tout en aillant un taux de livraison similaire.

On remarque que le routage opportuniste nécessite un mécanisme distribué d'élection du/des nœuds relayeurs qui consomme de l'énergie [Lampin et al., 2012] et qui peut mener à des réplications de paquets (qui conduit aussi à une surconsommation d'énergie). La fiabilité a donc un coût énergétique, nous commentons cet aspect dans la section 5.7 du Chapitre 5.

Dans le Chapitre 3 nous évaluons et comparons, à l'aide d'un modèle théorique, la fiabilité du routage classique et celle du routage opportuniste.

2.1.3 Solutions *cross-layer* : MAC et routage

Certains protocoles définissent à la fois des mécanismes d'accès au canal et de routage. Ces protocoles sont dits *cross-layer*. Cela permet d'éviter les problèmes de compatibilité qui peuvent apparaître quand les protocoles MAC et de routage sont conçus de manière totalement indépendantes. C'est le cas par exemple de RPL et 802.15.4 : RPL permet de faire du routage sur des DAG, mais au niveau MAC les structures sont des étoiles ou des réseaux maillés [Pavkovic et al., 2011]. De plus, la construction de ces structures MAC ne permet pas toujours d'optimiser les métriques visées au niveau routage.

1-hopMAC [Watteyne et al., 2006b] est un protocole MAC qui utilise une métrique de routage dans son fonctionnement. L'accès au canal se fait par contention. Un nœud qui a un paquet à émettre envoie d'abord une requête. Les nœuds qui reçoivent la requête vont répondre après un temps qui est proportionnel à la métrique de routage utilisée. Le nœud émetteur enverra ensuite le paquet de données au meilleur nœud qui a répondu à la requête. CMAC [Liu et al., 2009] utilise le même principe mais y ajoute la technique de préambule de XMAC. Une alternance de paquets de requête et

période de réponse composent le préambule. Les nœuds répondent en fonction de leur métrique durant les périodes de réponse. Ce type de solutions permet de départager les nœuds grâce à la métrique de routage. Cependant, si plusieurs nœuds dans un même voisinage possèdent la même valeur de métrique cela mène à des collisions durant la période de réponses aux requêtes.

D'autres approches, contrairement à celles présentées ici, permettent de garantir des délais bornés. Elles sont basées sur un ordonnancement global du réseau. Nous les détaillons dans la section 2.2.3.

2.2 Les communications temps-réel dans les réseaux de capteurs sans fil

Dans cette section, nous nous intéressons aux protocoles qui permettent l'acheminement des données dans les RCsF et qui visent à garantir une borne temporelle sur le délai de bout en bout. Comme mentionné dans le chapitre 1, les délais de bout en bout dans les RCsF proviennent essentiellement du temps d'accès au médium et de la longueur des routes. Pour pouvoir borner ces délais :

- au niveau MAC, il faut borner le temps d'accès au canal qui dépend du protocole utilisé, mais aussi du nombre de nœuds qui ont du trafic à écouler : borner l'accès au médium permet de borner le temps nécessaire pour faire un saut ;
- au niveau du routage, il faut pouvoir contraindre le nombre de sauts nécessaires pour atteindre le puits.

En plus de l'aspect temporel, les applications critiques nécessitent un niveau de fiabilité élevé. Dans les RCsF, cela se traduit par un taux de livraison important.

Toutes les solutions proposées dans la littérature ne donnent pas une borne stricte sur les délais et certaines ne prennent pas en compte les aspects large échelle et consommation d'énergie pourtant essentiels pour les RCsF.

2.2.1 Protocoles MAC temps-réel

Au niveau MAC pour borner les accès, les protocoles utilisent des mécanismes qui permettent d'éviter les collisions ou bien de les résoudre dans un temps borné. Les protocoles présentés assurent le respect de bornes temporelles sous hypothèse de liens fiables. Ils ont un degré de réactivité par rapport au trafic qui varie en fonction de la méthode d'accès.

Les solutions les plus statiques implémentent un accès TDMA (*Time Division Multiple Access*). Avec ce type d'accès, le temps est découpé en *slots* et chaque nœud dispose d'un ou plusieurs *slots* réservés. Dans ce cas, il est facile de déterminer le délai maximum d'un saut, c'est le délai du paquet du nœud qui est ordonnancé en dernier. Cependant, ce type de solution manque de réactivité. Si la plupart des nœuds n'ont pas de trafic mais qu'un nœud en a beaucoup, il devra patienter plusieurs *slots* alors que la plupart ne sont pas utilisés. De plus, les nœuds doivent se réveiller dans tous les *slots* de leurs voisins pour vérifier si un paquet est transmis.

FIGURE 2.11 – Mécanisme d'allocation de GTS de 802.15.4

Pour palier à ce problème, des solutions d'ordonnancement des accès aux *slots* en fonction du trafic de chaque nœud ont été proposées. On peut notamment citer 802.15.4. En effet, la période CFP de ce protocole permet de garantir des *slots* de temps aux nœuds. Comme représenté sur la Figure 2.11, les nœuds qui veulent du trafic garanti (nommé GTS pour *Guaranteed Time Slot*) font une demande dans la période CAP de la trame (la demande est noté *GTS-R*). Le coordinateur du réseau alloue ensuite les *slots* (dans la limite des *slots* disponibles) au nœud qui a fait la requête, la table d'allocation de la période CFP est décrite dans les *beacons*. Cette solution permet de s'adapter au trafic mais les requêtes sont faites de manière non déterministe, il est donc possible, en cas de collisions, que le nœud n'accède pas au médium avant l'échéance.

I-EDF [Caccamo et al., 2002] est un protocole synchrone basé sur l'algorithme d'ordonnancement *Early Deadline First* [Liu and Layland, 1973]. Son fonctionnement général est décrit par la figure 2.12. Cet algorithme consiste à ordonnancer en priorité les paquets dont l'échéance est la plus proche. Le réseau est découpé en cellules hexagonales. Les nœuds de chaque cellule forment un *cluster* qui opère sur une fréquence porteuse différente des cellules voisines. Le temps est divisé entre les communications intra-cellulaires et inter-cellulaires, comme représenté sur la Figure 2.12. Pour les communications intra-cellulaires, chaque nœud de la cellule connaît la périodicité et la taille des messages de tous les nœuds du *cluster*, ils peuvent donc calculer et appliquer le même ordonnancement en appliquant l'algorithme *Early Deadline First*, ce qui permet d'écouler le trafic sans collision. Entre deux communications intra-cellulaires, des *slots* de temps sont réservés pour les communications inter-cellulaires. Durant cette période, un nœud routeur qui est au centre de la cellule émet les données récupérées durant les communications intra-cellulaires. Un nœud routeur peut émettre dans 6 directions qui correspondent aux 6 côtés de la cellule hexagonale. Pour émettre dans une direction, il utilise la fréquence correspondant au voisin qu'il veut atteindre. Cette solution permet de garantir des bornes sur les délais. Cependant, la consommation d'énergie n'est pas prise en compte, il n'y a pas de cycle d'endormissement. De plus, la configuration en cellules hexagonales est contraignante, cela suppose que la topologie du réseau permette cette organisation avec le nœud routeur au centre de la cellule. Enfin, les nœuds doivent être capables de communiquer sur 7 fréquences porteuses différentes, cela augmente le coût des éléments du réseau ce qui peut être un problème pour les réseaux large échelle.

TRAMA [Rajendran et al., 2006] est un protocole MAC sans collision et efficace en énergie pour les RCsF. Les nœuds sont supposés synchronisés globalement. Le temps est divisé en *slots* avec une partie pour laquelle l'accès se fait par contention et une partie pour laquelle les accès sont déterministes. Il comprend une phase

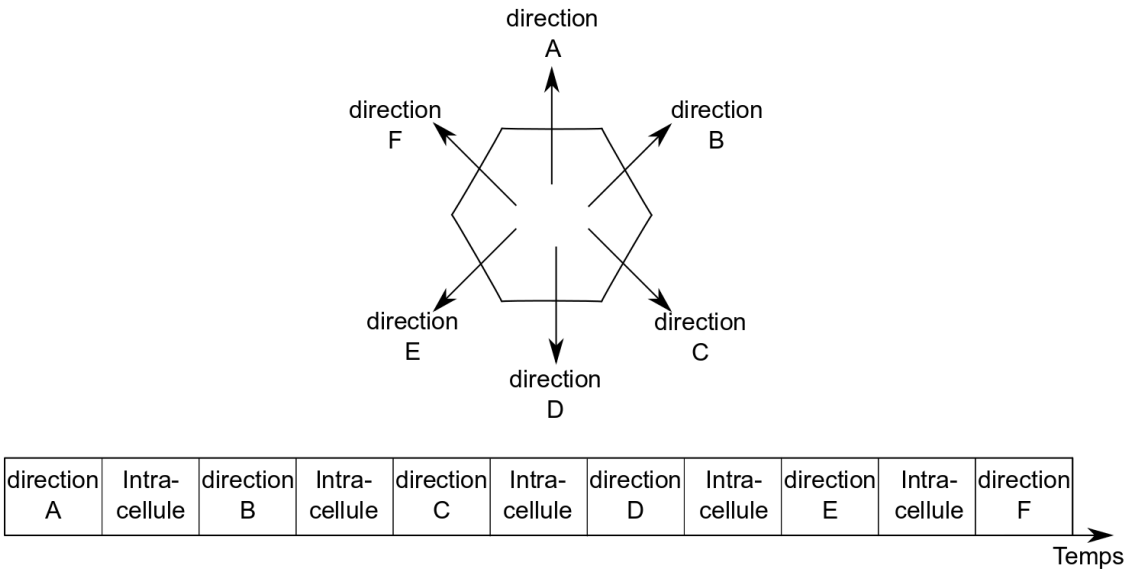


FIGURE 2.12 – Comportement général du protocole IEDF

de découverte du 2-voisinage réalisée durant une période de contention. Les nœuds échangent ensuite des informations sur le trafic qu'ils auront à émettre durant une période prédéfinie. Pour cette période, en fonction des informations de voisinage et de trafic, chaque nœud peut alors calculer sa priorité dans chaque *slot* de temps de la période sans contention. Le nœud qui a du trafic et qui est le plus prioritaire pour un *slot* peut l'utiliser. Les priorités des nœuds sont connues grâce à la découverte du 2-voisinage, elles sont permutées grâce à un générateur pseudo-aléatoire suivant la technique décrite dans [Bao and Garcia-Luna-Aceves, 2001], cela permet de garantir une équité dans les accès au canal. Les principaux inconvénients de TRAMA sont la lourdeur des échanges de contrôle pour indiquer le trafic et la topologie et le fait que les informations des nœuds doivent être cohérentes pour garantir l'absence de collisions dans les *slots* en accès fixe. En effet, des pertes de paquets d'informations de trafic ou des collisions lors de la découverte du 2-voisinage peuvent mener à des incohérences entre les nœuds. Il existe donc un risque que deux nœuds du même 2-voisinage s'attribuent la plus grande priorité pour le même *slot*. Enfin si le réseau est très dense, le 2-voisinage d'un nœud peut atteindre une taille trop importante pour être stockée dans sa mémoire vive. On note que LEMMA [Macedo et al., 2009] reprend les principes de TRAMA mais en chargeant un nœud en particulier d'allouer les *slots* pour une partie de ses voisins (ses enfants dans l'arbre de routage). L'allocation des *slots* est réalisée avec des accès au canal aléatoires et ne permet pas de borner strictement le délai de bout en bout (les auteurs fournissent un délai théorique moyen).

f-MAC [Roedig et al., 2006] se distingue des protocoles précédents car il n'y a pas de découpage du temps en *slots* ou en trames. f-MAC est un protocole asynchrone et localisé qui permet un accès déterministe au médium. Un nœud qui doit émettre un paquet envoie plusieurs instances du même paquet (nommé *framelet*) avec une période

unique dans le 2-voisinage. Les auteurs montrent qu'en appliquant des règles mathématiques pour le choix des périodes, on peut garantir qu'au moins une *framelet* de chaque nœud est transmise sans collision. Une borne sur le délai de transmission d'un paquet est fournie dans [Roedig et al., 2006]. Cependant, elle augmente exponentiellement avec le nombre de nœuds dans un 2-voisinage. De plus, l'information utile est répétée de nombreuses fois sur le canal, cela réduit le débit utile. Le fonctionnement de f-MAC est totalement asynchrone et il n'y a pas de mécanisme d'échantillonnage de préambule. Les nœuds qui n'ont rien à transmettre écoutent continuellement le canal ce qui entraîne une consommation d'énergie importante. Les auteurs affirment qu'il est possible d'utiliser un mécanisme d'échantillonnage de préambule avec f-MAC. Dans ce cas, le mécanisme des périodes uniques perd de son intérêt. En effet, avec de l'échantillonnage de préambule, les nœuds d'un même voisinage sont synchronisés à la fin du préambule. Les nœuds ayant une période unique, cet identifiant peut être utilisé pour envoyer le message à un instant unique, évitant les collisions et rendant inutile la répétition du message. Dans ce cas, f-MAC est équivalent à un TDMA avec échantillonnage de préambule.

Dual-Mode MAC [Watteyne et al., 2006a] est une couche MAC temps-réel dur pour RCsF linéaires, c'est-à-dire quelle n'est applicable que pour des réseaux qui ont une topologie en ligne (surveillance d'autoroutes, de pipelines, etc). Le protocole a deux modes d'opération. Un mode non protégé qui permet un temps d'accès au médium quasi-optimal quand il n'y a pas de collision. Quand une collision survient, le protocole passe en mode protégé : un chemin est alors réservé pour garantir un délai borné. Quand le nombre de paquets reçu par le puits retombe en dessous d'un seuil, le protocole repasse en mode non protégé. Les deux modes permettent un bon compromis entre performances et garantie du délai de bout en bout. Les auteurs définissent analytiquement le pire délai. De plus, le protocole est modélisé et vérifié formellement avec le *model checker* UPPAAL. Cependant, le protocole fonctionne seulement sur des réseaux qui ont une topologie ligne ce qui réduit grandement les applications. De plus, la consommation d'énergie des nœuds n'est pas prise en compte.

2.2.2 Protocoles de routage temps-réel

Pour pouvoir borner les délais de bout en bout, il faut être capable de borner le nombre de sauts pour atteindre le puits. Cependant, ce n'est pourtant pas l'approche adoptée par la plupart des solutions de la littérature. Ces solutions se présentent comme temps-réel non strict (*soft real-time*) et sont basées sur les coordonnées géographiques des nœuds.

SPEED [Stankovic and Abdelzaher, 2003] se base sur les coordonnées géographiques et du routage *greedy*. Chaque nœud maintient une table contenant une entrée pour chacun de ses voisins avec une métrique qui représente leur vélocité. La vélocité d'un voisin est calculée en divisant l'avance géographique qu'il procure dans la direction du puits par le délai pour envoyer le paquet à ce voisin. C'est cette métrique qui est utilisée par le mécanisme *greedy*. Un voisin est choisi s'il a une vélocité suffisante pour satisfaire l'échéance du paquet à envoyer. Cependant, aucune garantie sur le respect des échéances n'est fournie. Le problème principal vient de l'utilisation de la

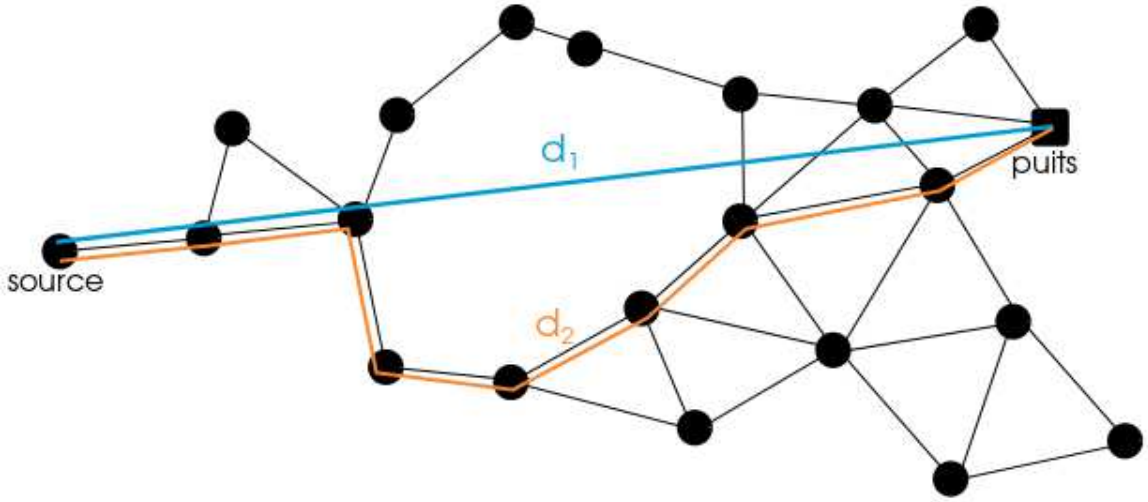


FIGURE 2.13 – Problème dans le calcul de vélocité : la distance estimée d_1 est plus petite que celle qui est réellement parcourue d_2.

distance géographique pour estimer la distance qu'un paquet va parcourir et donc sa vélocité. Même si cela donne une bonne estimation dans les réseaux denses, s'il y a des trous dans le réseau la distance réelle parcourue est plus longue que prévue. Cet effet est décrit par la figure 2.13, sur laquelle on peut observer la différence entre la distance estimée d_1 et celle qui est vraiment parcourue par le paquet d_2. SPEED propose tout de même un mécanisme de détection de congestion qui limite les pertes de paquets.

MM-SPEED [Felemban et al., 2006] étend SPEED en ajoutant la possibilité d'avoir plusieurs niveaux de services. Le protocole reprend la vélocité de SPEED et propose de définir plusieurs réseaux virtuels qui peuvent fournir des vélocités différentes (plusieurs instances de SPEED). Les auteurs s'intéressent aussi à la fiabilité des communications, ils définissent une métrique de fiabilité qui permet de prendre des décisions locales sur la meilleure route à suivre. Les paquets qui requièrent une très grande fiabilité peuvent être envoyés par plusieurs chemins (*multi-path*). MM-SPEED améliore les performances de SPEED, mais souffre du même problème : l'impossibilité de garantir strictement un délai de bout en bout.

RPAR [Chipara et al., 2006] reprend aussi la métrique de vélocité de SPEED. Les auteurs décrivent le compromis entre l'énergie et le délai de bout en bout. En effet, pour réduire le délai, il faut faire moins de sauts car chaque retransmission prend un temps incompressible. Pour réaliser moins de sauts, une solution est d'augmenter la puissance de transmission des nœuds. Cela à deux effets : l'énergie consommée augmente et le rayon d'interférence des nœuds aussi, diminuant ainsi le nombre de communications concurrentes possibles dans le réseau. Les auteurs considèrent que la perte de capacité n'est un problème que lorsque le réseau atteint sa limite de capacité et que le respect des échéances des paquets est plus important que la capacité. RPAR choisit le prochain saut en fonction de la vélocité et de l'énergie. Le voisin choisi est celui qui consomme le moins d'énergie parmi ceux qui respectent la vélocité requise par

le paquet. Si aucun voisin ne respecte le délai, un des voisins est élu pour augmenter sa puissance de transmission. RPAR souffre des mêmes problèmes que SPEED et MM-SPEED.

Le problème principal des protocoles de routage présentés ici est que l'estimation de la vélocité requise par les paquets pour respecter leur échéance est fonction de la distance géométrique. La topologie du réseau peut présenter des trous, ce qui augmente cette distance et donc les délais. Ce problème peut être résolu en utilisant un routage par gradient. En effet, avec un protocole comme GRAB, le nombre de sauts qui sépare un nœud du puits est connu. Il est donc possible de borner le nombre de sauts maximum dans le réseau. En utilisant une couche MAC qui borne le délai d'un saut, on peut garantir le délai de bout en bout si on fait l'hypothèse que les liens sont fiables. En réalité, les liens radio sont non-fiables, il faut donc prendre en compte cet aspect. Ce point est détaillé dans le Chapitre 3.

2.2.3 Approches *cross-layer* pour les communications temps-réel des RCsF

Comme mentionné dans la section 2.1.3, certaines couches MAC et routages sont difficiles à interfacer. Cela rend le comportement global du système dur à prévoir, ce qui est problématique lorsqu'il faut fournir les garanties requises par les applications critiques. Pour pallier à ce problème, des protocoles implémentant les couches MAC et routages, dits *cross-layers*, ont été proposés.

Une première approche consiste à produire un ordonnancement des communications dans le réseau qui combine MAC et routage, seulement quand un événement est détecté et uniquement pour cet événement. PR-MAC [Chen et al., 2007] implémente ce type de solution. C'est un protocole synchrone temps-réel pour RCsF. Il définit les couches MAC et routage. Le but de ce protocole est de détecter un événement, puis de surveiller la zone où a eu lieu l'événement. Quand un événement est détecté, une alarme est envoyée au puits. Le puits répond avec un paquet qui réserve un chemin pour le *monitoring* périodique. Le chemin est réservé pour une fréquence donnée. Chaque événement est associé à une fréquence porteuse, cela permet de ne pas avoir de collisions entre les différents trafics dans le réseau. Le *monitoring* est temps-réel, des *slots* sont réservés pour remonter le trafic au puits. Cependant, la détection de l'événement ainsi que la création du chemin de *monitoring* ne sont pas réalisées dans un délai borné. De plus, il faut qu'il y ait au moins autant de canaux radios que d'événements pour que le protocole fonctionne.

Une seconde approche consiste produire un ordonnancement global du réseau. Pour cela, la topologie du réseau est récupérée par le puits. Il se charge de produire un ordonnancement qui permet de remonter le trafic de chaque nœud vers le puits. Cet ordonnancement a une durée bornée, le délai de bout en bout est donc borné par la taille de l'ordonnancement. Cette approche est utilisée par TSMP [Doherty and Pister, 2008] et PEDAMACS [Ergen and Varaiya, 2006]. TSMP [Doherty and Pister, 2008] est un protocole qui fournit un ordonnancement global du réseau. Ce protocole utilise un accès au canal de type MF-TDMA (pour *Multi-*

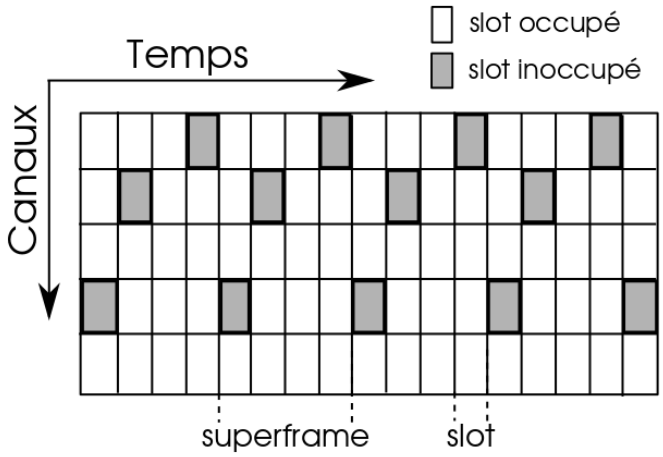


FIGURE 2.14 – Illustration de l'espace temps-canaux de TSMP avec une *superframe* de 4 *slots* et 5 canaux

Frequency TDMA) comme illustré par la Figure 2.14. L'algorithme d'ordonnancement n'est pas détaillé dans [Doherty and Pister, 2008], mais l'ordonnancement produit a une durée finie et est appelé *superframe*. Il est donc possible de borner le délai de bout en bout par la taille d'une *superframe*. TSMP fournit aussi un mécanisme d'acquittement des paquets au niveau MAC avec la possibilité de les retransmettre. PEDAMACS [Ergen and Varaiya, 2006] a le même principe de fonctionnement que TSMP, mais un seul canal radio est utilisé. Les auteurs décrivent les mécanismes de récupération de la topologie et de production de l'ordonnancement par le puits. Les nœuds disposent de plusieurs puissances de transmission : le puits peut atteindre tous les nœuds du réseau, les autres nœuds disposent de deux puissances de transmission, une pour communiquer avec leurs voisins et une pour identifier leurs interférants. Une synchronisation du réseau est réalisée grâce à des paquets de synchronisation émis périodiquement par le puits.

Le protocole fonctionne en trois phases. Lors de la première phase, les nœuds découvrent leurs voisins et leurs interférents. Pour cela, ils utilisent leurs deux puissances de transmissions successivement. Les informations récupérées durant la première phase sont envoyées au puits durant la deuxième phase. Durant ces deux phases, les nœuds accèdent au canal de manière aléatoire, le puits récupère donc une vue de la topologie du réseau et, à partir de cette information, il produit un ordonnancement global du réseau qui garantit que chaque nœud peut envoyer un paquet de données au puits. Les auteurs donnent, pour différents types de réseaux (arbres avec des hypothèses sur les interférences), la borne temporelle garantie. La troisième phase consiste à l'exécution de l'ordonnancement par les nœuds (que le puits envoie en *broadcast* avec sa puissance de transmission maximale). Chaque nœud se réveille seulement pour recevoir et pour envoyer des paquets dans les *slots* qui lui sont réservés.

On note que RT-LINK [Rowe et al., 2006] reprend les mêmes principes que PEDAMACS, mais permet d'ajouter des nœuds dans l'ordonnancement durant la phase de transmission des paquets de données. Pour cela, une partie de la trame est prévue

pour envoyer des paquets de demande de *slots*. Ces demandes se font en CSMA/CA et sont relayées de cette façon vers le puits, qui modifie ensuite l'ordonnancement pour intégrer les nouveaux nœuds.

Les solutions centralisées permettent de fournir une garantie sur le délai de bout en bout des paquets, sous hypothèse de liens radio fiables. En réalité, comme on le verra dans le Chapitre 5, les pertes de paquets sont problématiques pour ces solutions. De plus, leur principal inconvénient est le passage à l'échelle. En effet, le puits doit disposer d'une vue cohérente du réseau. Dans le cas de réseaux larges échelles, il est difficile de récupérer toute la topologie. Pour PEDAMACS, l'hypothèse que le puits peut communiquer avec tous les nœuds est contraignante. Enfin, ces solutions sont adaptées à un trafic périodique, ou qui change peu souvent au cours du temps. Dans le cas contraire, il faudrait reconstruire un ordonnancement fréquemment ce qui est coûteux en énergie.

2.3 Conclusion

Les solutions d'acheminement des données de la littérature ne permettent pas, pour la plupart, de garantir des bornes temporelles sur le délai de bout en bout. C'est en partie dû au fait que les solutions proposées se sont surtout focalisées sur la consommation d'énergie et l'auto-organisation. Au niveau MAC, la plupart des solutions proposées définissent un accès aléatoire au médium, cela ne permet pas de borner le temps d'un saut. Au niveau routage, les solutions géographiques, largement utilisées car passant l'échelle, ne permettent pas de borner la longueur des chemins.

Parmi les solutions temps-réel de la littérature, certaines [Caccamo et al., 2002] [Roedig et al., 2006] [Ergen and Varaiya, 2006] [Doherty and Pister, 2008], conçues avec une approche systèmes temps-réel, permettent effectivement de respecter des contraintes temporelles mais ne prennent pas suffisamment en compte les aspects propres à ces réseaux (passage à l'échelle, énergie, etc). Par exemple, les ordonnancements globaux permettent de maîtriser le délai de bout en bout tant au niveau MAC que routage. Cependant, il y a un problème de passage à l'échelle avec ces solutions. D'autres solutions, conçues avec une approche réseau, prennent en compte les caractéristiques des RCsF mais ne permettent pas de garantir les délais de bout en bout. C'est le cas de SPEED et ses dérivés.

Une solution satisfaisante doit pouvoir, au niveau MAC, prioriser les accès de chaque nœud d'un même domaine d'interférence de manière déterministe et locale pour pouvoir être prédictible et passer l'échelle. Elle doit permettre un accès dynamique au canal, laissant les nœuds qui ont du trafic y accéder tandis que les autres peuvent rester endormis pour économiser l'énergie. Les décisions de routage doivent se prendre de manière locale pour passer l'échelle, permettre de pouvoir borner le nombre de sauts qu'un paquet doit effectuer pour atteindre le puits pour borner les délais de bout en bout et maximiser la fiabilité des transmissions de bout en bout pour répondre aux besoins de fiabilité des applications critiques. Dans les chapitres 4 et 5 nous nous attachons à construire et évaluer une telle solution.

Dans le chapitre suivant nous étudions la fiabilité des transmissions de bout en bout dans les RCsF, nous nous attachons notamment à évaluer la fiabilité des routages classiques et opportunistes pour conclure quant à la méthode la plus adaptée dans le cadre des applications critiques.

Chapitre 3

Fiabilité des transmissions de bout en bout dans les RCsF

Dans les chapitres précédents nous avons présenté les difficultés que représente l'acheminement de données en temps-réel dans les RCsF. De plus, nous avons évoqué à plusieurs reprises le problème de la fiabilité de l'acheminement de données. En effet, un protocole qui permet de garantir des bornes temporelles strictes doit avoir un taux de livraison suffisamment élevé pour l'application visée. Il perd sa valeur si la plupart des informations, qui sont critiques, n'arrivent pas au puits. Dans ce chapitre, nous étudions en détails la fiabilité qu'il est possible d'obtenir dans un RCsF avec des liens radios probabilistes. Nous définissons la fiabilité comme la probabilité de succès d'une transmission de bout en bout. Nous proposons une étude théorique qui a pour originalité de prendre en compte la longueur des liens dans le calcul des probabilités de réception de paquets et aussi de permettre d'étudier la fiabilité du routage de type opportuniste. Notre démarche est la suivante :

- Nous présentons d'abord les phénomènes physiques qui rendent le lien radio non-fiable.
- Nous détaillons ensuite les outils nécessaires à cette étude :
  - Des modèles de lien radio : un simplifié et un réaliste. De ces deux modèles, nous déduisons la probabilité de réception d'un paquet (avec et sans retransmission). Pour le modèle réaliste, elle dépend de la distance émetteur-récepteur, contrairement au modèle simplifié. Ces probabilités sont ensuite injectées dans les équations de fiabilité de bout en bout.
  - Un modèle de topologie : nous définissons la forme de l'aire contenant le réseau et la répartition des nœuds. On peut noter que l'on ne considère pas de topologie arbitraire (les positions exactes des nœuds ne sont pas connues).
- Nous prenons des hypothèses sur les protocoles : nous présentons les approches de routage opportuniste et classique. Au niveau MAC, nous différencions les protocoles déterministes et probabilistes (avec une probabilité de perte de paquet).
- Nous utilisons ces outils et hypothèses pour évaluer la fiabilité des approches de routage opportuniste et classique avec les deux modèles de lien radio et les hypothèses sur les couches MAC.

• Nous vérifions les résultats théoriques par simulation.

Les résultats mettent en évidence que le routage opportuniste est plus fiable que le routage classique. Ce type routage n'est cependant pas utilisé par les protocoles critiques temps-réel de la littérature. On observe que le routage opportuniste souffre d'un goulot d'étranglement de fiabilité au niveau du puits : le dernier saut est le moins fiable (on définit plus précisément cette notion dans la section 3.3.2). On conclut que les performances de ce type de routage peuvent être améliorées de manière non négligeable par l'ajout d'un deuxième puits accolé au premier, et ce pour un coût relativement faible : le réseau ne change pas, on doit seulement ajouter un nœud et récupérer les données collectées par deux puits au lieu d'un seul.

Nous commençons, dans la section suivante, par un état de l'art des modèles théoriques de fiabilité des RCsF.

3.1 État de l'art sur les modèles de fiabilité des RCsF

Plusieurs modèles de fiabilité pour RCsF ont été proposés dans la littérature dans lesquels la fiabilité est définie en fonction de différents critères : le débit au puits [AboElFotoh et al., 2006], la connectivité [Zhang and Gorce, 2007] ou la détection d'événements [Shaikh et al., 2007] [Caleffi et al., 2008].

Dans [AboElFotoh et al., 2006], les auteurs définissent la fiabilité comme étant la probabilité d'avoir un débit minimum au niveau du puits. Les auteurs proposent un algorithme pour déterminer la fiabilité d'un RCsF arbitraire, c'est-à-dire que le graphe du réseau est fourni en entrée de l'algorithme. Un nœud peut être opérationnel ou non avec une certaine probabilité associée. A partir des états de chaque nœud, l'état du réseau est déterminé. L'algorithme consiste alors à explorer tous les états du système et à sommer les probabilités des états où il existe un arbre depuis les sources vers le puits et où le débit est assez important. Cette approche est centrée sur les défaillances provenant des nœuds et non des transmissions. De plus, les auteurs considèrent un débit minimum alors que nous nous intéressons à la fiabilité des transmissions de bout en bout pour un paquet donné (typiquement, une alarme remontant vers le puits). Enfin, cette méthode prend en entrée un réseau arbitraire (graphe du réseau connu), ce qui n'est pas le cas de notre méthode qui considère des réseaux aléatoires (on dispose seulement de la distribution de probabilité concernant les positions des nœuds).

Dans [Zhang and Gorce, 2007], les auteurs étudient la connectivité des RCsF avec des liens non-fiables. Ils considèrent un modèle réaliste où la probabilité de réception d'un paquet dépend de la distance émetteur-récepteur. Ils évaluent le degré moyen d'un nœud pour différents modèles de propagation (avec et sans *fading*) et montrent que le phénomène de *fading* fait croître le degré moyen grâce à l'utilisation des liens non-fiables longs. Ce constat est notamment mis en pratique dans [Lampin et al., 2012] (présenté dans le chapitre précédent) qui tire parti de ces liens (nommées liens opportunistes) pour améliorer les performances de taux de livraison

et de délai de bout en bout. Notre étude, plutôt que de s'intéresser au degré moyen d'un nœud comme dans [Zhang and Gorce, 2007], se concentre sur la probabilité de succès des transmissions de bout en bout dans le cas où les liens opportunistes sont utilisés (routage opportuniste) et dans le cas où il ne le sont pas (routage classique).

Dans [Shaikh et al., 2007], un modèle pour calculer la fiabilité au niveau transport dans les RCsF est proposé : c'est la probabilité qu'un phénomène physique soit détecté par le puits, la fiabilité des protocoles de routage, et donc de la transmission de bout en bout d'un paquet, est donnée en entrée. Il est basé sur une représentation en schéma-bloc où chaque bloc représente une opération du réseau. Les auteurs incluent beaucoup de possibilités de fautes : pertes dues aux transmissions, destructions de nœuds, mauvais comportement des capteurs, batterie épuisée, faute passagère (typiquement résolue par un redémarrage du nœud). Cependant, ils ne fournissent pas de modèle statistique accompagnant ces types de fautes. Les fiabilités de protocoles de transport sont comparées, mais les auteurs ne détaillent pas les équations qui permettent d'obtenir la probabilité de succès d'une transmission de bout en bout (ils se placent au niveau transport). C'est au contraire un paramètre d'entrée de leur étude, la fiabilité étant fonction du nombre de sources qui détectent l'événement. Ce modèle est donc assez "haut niveau" et ne permet pas de représenter la complexité de comportement des liens radios et son impact sur la fiabilité des transmissions de bout en bout. En fait, notre étude de fiabilité, plus "bas niveau", peut être considérée comme une entrée pour leur modèle.

Les auteurs de [Caleffi et al., 2008] proposent un modèle qui permet d'évaluer la fiabilité de protocoles de routages multi-chemins pour les réseaux MANET. Les auteurs ont une approche assez similaire à celle développée dans ce chapitre : ils calculent la probabilité de succès d'une transmission bout en bout à partir de la probabilité de réussite d'un saut. Ils s'intéressent à la qualité du lien radio, mais dans leurs résultats, ils considèrent que cette qualité est uniforme dans le réseau, c'est un paramètre d'entrée de leur étude. Nous allons plus loin en incluant le fait que la qualité du lien radio (probabilité de réception) dépend de la distance émetteur-récepteur. De plus, leur méthode s'applique à des réseaux arbitraires, alors que nous prenons en compte des réseaux aléatoires. Enfin, leur modèle considère des transmissions point-à-point là ou nous nous concentrons sur des transmissions *convergecast*.

Notre proposition est complémentaire à ces modèles. Elle permet de prendre en compte un modèle de canal où la probabilité de réception dépend de la distance émetteur-récepteur et d'étudier le routage opportuniste (qui n'est étudié dans aucun des articles cités). De plus, nous nous plaçons dans un cas plus général que les modèles présentés en étudiant des réseaux aléatoires et non arbitraires.

3.2 Modèles de RCsF

Dans cette section, nous présentons les outils et les hypothèses qui nous permettent de réaliser l'étude de fiabilité des transmissions de bout en bout. Nous décrivons d'abord les phénomènes qui rendent le lien radio non-fiable. Nous détaillons ensuite

les modèles de liens utilisés dans l'étude, puis les hypothèses sur la topologie du réseau, et enfin sur les types de protocoles.

3.2.1 Modèle de lien radio

Dans cette section, nous présentons d'abord des propriétés clefs du le lien radio : nous décrivons les phénomènes physiques qui rendent le lien radio non-fiable. Nous présentons ensuite les modèles de liens utilisés dans l'étude de fiabilité des transmissions de bout en bout : un modèle simplifié et un modèle réaliste.

Généralités sur le lien radio

Les transmissions radios sont soumises à des phénomènes physiques (atténuation, bruit, réflexions, diffractions, etc) qui perturbent la réception des paquets. La capacité d'un nœud à décoder un message dépend de deux éléments : la puissance du signal utile reçu et la puissance des autres signaux [Goldsmith, 2005] de l'environnement. Les erreurs de réceptions de messages dépendent de l'intensité du signal utile qui est atténué par les pertes en espace libre et le phénomène de *fading* ; et de l'intensité signaux non-utiles : le bruit de fond et les interférences dues aux transmissions simultanées [Goldsmith, 2005]. Nous décrivons ces phénomènes dans la suite de cette section.

Le phénomène de pertes en espace libre exprime le fait que la puissance d'un signal émis dans l'espace se réparti sur la surface d'une sphère, la puissance reçue en un point décroît donc de manière quadratique en fonction de la distance à l'émetteur [Goldsmith, 2005] :

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi d}\right)^2 \tag{3.1}$$

avec P_r la puissance du signal reçu, P_t la puissance du signal transmis, G_t et G_r les gains des antennes de transmission et réception, λ la longueur d'onde de la fréquence d'émission et d la distance émetteur-récepteur.

En plus de cette atténuation en espace libre, les phénomènes de réflexion et diffraction sur des objets de l'environnement introduisent des variations de la puissance reçue. Ces atténuations ne sont pas prédictibles car liées à des phénomènes aléatoires [Goldsmith, 2005] (mouvements d'objets, de personnes, du récepteur, etc). Ce phénomène d'atténuations aléatoires est nommé *fading* [Goldsmith, 2005] et est modélisé par l'ajout d'une composante aléatoire à l'atténuation du signal. La Figure 3.1 représente un exemple de formes de contours de puissance reçue constante dans les cas où l'on ne prend en compte que les pertes en espace libre (ligne continue) et les pertes en espace libre plus une composante de *fading* (ligne pointillée). Dans le premier cas la portée radio (qui est définie par un seuil de puissance reçue) est un disque, dans le second cas la portée radio a une forme aléatoire.

On considère maintenant les signaux qui perturbent la réception des signaux utiles : le bruit de fond et les interférences. Le bruit de fond est le signal présent dans le système radio quand aucun signal utile n'est reçu, il est principalement dû au bruit

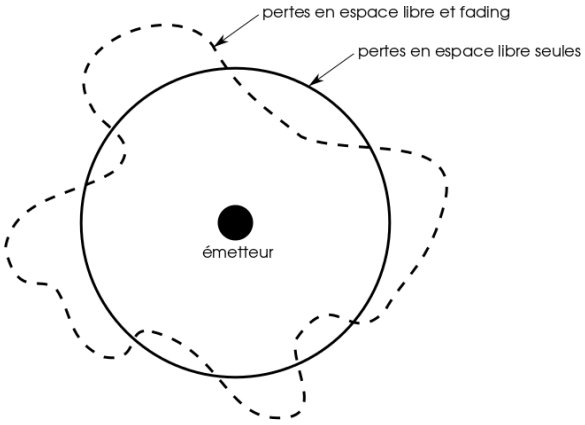


FIGURE 3.1 – Contours de puissance constante pour les pertes en espace libre seules et les pertes en espace libre avec une composante de *fading*

thermique [Goldsmith, 2005]. Le bruit thermique est décrit par une variation de la tension aux bornes d'une résistance qui provient de l'agitation thermique des électrons dans la résistance. Cette variation de tension apparaît dans les circuits électroniques radios et perturbent le signal utile (données) à la réception [Goldsmith, 2005].

Une autre source d'erreurs à la réception des paquets est les interférences dues aux transmissions simultanées : elles correspondent aux signaux qui ne sont pas destinés à un nœud, mais qui sont quand même reçus. Ces signaux s'ajoutent au bruit thermique et perturbent la réception de signaux utiles (adressés au nœud). On note que le but d'une couche MAC est précisément de maintenir le niveau d'interférences assez bas pour minimiser les erreurs de réceptions de paquets dues aux interférences (aussi appelées collisions) et de pouvoir résoudre ces collisions lorsqu'elles apparaissent.

A partir des éléments présentés ci-dessus, on peut définir le rapport de signal à bruit et interférences ou SINR (pour *Signal to Interference plus Noise Ratio*) :

$$\gamma = \frac{P_r}{N_0.B + \sum_{i \in T} P_{r_i}} \tag{3.2}$$

avec P_r la puissance du signal utile reçu, N_0 est la densité de bruit, B la bande passante, T l'ensemble des nœuds interférants qui transmettent un signal, P_{r_i} la puissance du signal reçu du nœud interférant i. A partir du SINR et du schéma de modulation et codage utilisé, on peut déterminer le taux d'erreur bit (BER) ainsi que le taux d'erreurs paquet (PER) [Goldsmith, 2005]. Par exemple pour une modulation BPSK [Goldsmith, 2005] sans codage, le BER est donné par [Goldsmith, 2005] :

$$BER_{bpsk}(\gamma) = \frac{1}{2} erfc(\sqrt{\gamma}) \tag{3.3}$$

avec *erfc* la fonction d'erreur de Gauss [Goldsmith, 2005].

Dans la description du modèle de lien réaliste, nous détaillons comment sont pris en compte ces phénomènes physiques pour calculer la probabilité de réception d'un paquet en fonction de la distance émetteur-récepteur.

Modèle de lien simplifié

Dans l'étude de fiabilité que nous proposons, nous utilisons deux modèles de liens radios. Un modèle simplifié, dans lequel chaque nœud peut communiquer avec un ensemble prédéfini de voisins. Les transmissions sont cependant non-fiables. Un voisin a une probabilité P_{bcr} de recevoir le paquet. Dans le modèle simplifié, nous supposons que cette probabilité est la même pour tous les voisins indépendamment de leurs positions et est donnée en entrée du modèle. Ce modèle simplifié nous permet d'avoir une première approche pour effectuer une évaluation préliminaire des schémas de routage classiques et opportunistes. Nous allons ensuite plus loin en utilisant un modèle de lien réaliste basé sur le modèle de propagation *log-normal shadowing* présenté ci-après.

Modèle de lien réaliste

Le modèle de lien réaliste que nous employons est similaire à celui utilisé dans [Zhang and Gorce, 2007], il est basé sur le modèle de propagation *log-normal shadowing*.

Pour ce modèle, nous supposons que l'on peut déterminer un seuil N_{th} qui majore le bruit thermique et les interférences des transmissions concurrentes : $N_{th} \geq N_0.B + \sum_{i \in T} P_{r_i}$. Nous considérons que les protocoles de communications utilisés (principalement au niveau MAC) doivent être capables de maintenir le niveau d'interférences en dessous de ce seuil. Cela revient à considérer que la couche MAC est chargée d'empêcher les collisions (nous nous intéressons tout de même aux collisions dans la section 3.3.3). Sous cette hypothèse, ce sont les atténuations aléatoires du signal qui font varier le SINR et donc la probabilité de recevoir correctement (décoder) un paquet.

Pour représenter les atténuations aléatoires du signal, nous choisissons comme modèle de propagation le *log-normal shadowing*. Ce modèle permet de représenter le *shadowing* (ou *fading* lent), c'est-à-dire l'atténuation du signal radio par des objets mouvants. Ce modèle est considéré comme réaliste pour les RCsF [Zuniga and Krishnamachari, 2004]. Nous détaillons comment à partir du modèle de propagation, nous obtenons la probabilité qu'un paquet soit correctement reçu en fonction de la distance entre l'émetteur et le récepteur.

L'Equation 3.4 décrit l'affaiblissement en décibels du signal dû à la propagation.

$$PL_{dB}(d) = PL_{dB}(d_0) + 10\alpha \log_{10}(\frac{d}{d_0}) + X_\sigma \tag{3.4}$$

d_0 est une distance de référence, α est l'exposant d'affaiblissement, d est la distance entre l'émetteur et le récepteur et X_σ est une variable gaussienne en dB de moyenne 0 et d'écart-type σ. C'est cette variable gaussienne qui modélise l'aspect aléatoire de l'environnement.

Nous définissons la probabilité de succès d'une transmission comme la probabilité qu'un paquet envoyé par l'émetteur soit reçu par le récepteur en fonction de d la distance émetteur-récepteur. C'est à dire comme le complément du taux d'erreur paquet (PER : *Packet Error Rate*).

Symbole	Description	Valeur
N_p	Taille du paquet en bits	800
σ	Ecart type en dB	4
α	Exposant d'affaiblissement	2
N_{th}	Seuil de niveau de bruit et interférences en dBm	-110
f	Fréquence de la porteuse en MHz	868
B	Bande passante en kbps	500

TABLE 3.1 – Paramètres du modèle

$$P_{cr}(d) = 1 - PER \tag{3.5}$$

Le PER est défini en fonction du taux d'erreurs bit (sous l'hypothèse d'erreurs bit indépendantes) :

$$PER = 1 - (1 - BER)^{N_p} \tag{3.6}$$

où N_p est la taille du paquet en bits. Le BER dépend de la modulation utilisée et du *fading* sur le canal. Dans cette étude nous utilisons une modulation BPSK qui est une des alternatives proposées dans le standard 802.15.4 [IEEE, 2011b]. Comme mentionné précédemment, nous considérons un *fading* lent. Dans ce cas, l'expression du PER peut s'écrire [Proakis and Salchi, 2008] :

$$PER = 1 - \int_{\gamma=0}^{\infty} (1 - BER_{bpsk}(\gamma))^{N_p} . \frac{1}{\sigma\sqrt{2\pi}} . e^{-\frac{(\gamma-\bar{\gamma})^2}{2\sigma^2}} \tag{3.7}$$

avec BER_{bpsk} donné par l'Equation 3.3 et $\bar{\gamma} = 10.\log_{10}\left(\left(\frac{\lambda}{4\pi}\right)^2 \frac{P_t.d^{-\alpha}}{N_{th}}\right)$ le SINR moyen (les paramètres sont décrits dans le Tableau 3.1). L'Equation 3.7 lie donc le PER avec le SINR moyen qui est lui-même fonction de la distance émetteur-récepteur. Grâce à l'Equation 3.5, nous obtenons la probabilité de recevoir correctement un paquet en fonction de la distance émetteur-récepteur. Pour l'implémentation de l'Equation 3.5 (pour obtenir des résultats numériques des sections suivantes de ce chapitre), nous utilisons une approximation de l'Equation 3.7 développée dans [Ferrand et al., 2013]. Notons qu'avec ce modèle, les probabilités de réceptions sur les liens sont indépendantes et symétriques (pour deux nœuds A et B la probabilité de réception d'un paquet de A par B est égale à la probabilité de réception d'un paquet de B par A) : ces aspects ne sont pas toujours vérifiés expérimentalement [Heurtefeux et al., 2012] et ils seront commentés dans la section 3.5.

La Figure 3.2 est une représentation graphique de l'Equation 3.5. P_{cr} est tracé en fonction de la distance émetteur-récepteur pour plusieurs puissances de transmission (10mW, 1mW et 0.5mW). Les valeurs utilisées pour les autres paramètres sont données par le Tableau 3.1. On distingue trois zones : la zone connectée où la probabilité est très proche 1, la zone de transition où la probabilité est entre 0 et 1 et la zone déconnectée où la probabilité est très proche de 0. La largeur de ces zones dépend de la puissance de transmission. La largeur de la zone connectée et de celle de transi-

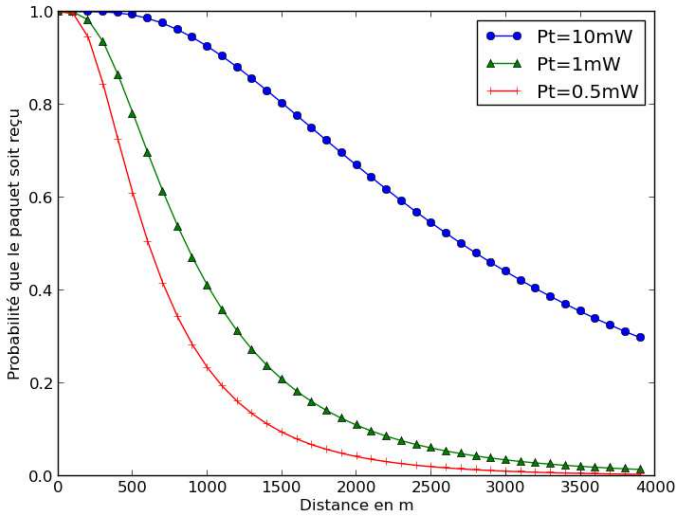


FIGURE 3.2 – Probabilité qu'un paquet soit correctement reçu en fonction de la distance émetteur-récepteur

tion augmentent avec cette puissance. Une zone de transition courte est intéressante, car on obtient un réseau où les nœuds sont soit connectés soit déconnectés sans intermédiaires, ce type de réseau a donc un comportement prédictible. Cependant, en pratique, on n'obtient pas ce type de réseau, car la zone de transition reste importante par rapport à la zone connectée.

Probabilité de succès avec retransmissions

Souvent, les protocoles définissent des mécanismes de retransmission pour augmenter la fiabilité des transmissions. Pour pouvoir prendre en compte ce type de comportement nous définissons la probabilité de recevoir le paquet correctement après au plus K retransmissions :

$$\begin{aligned} P_{cr\_ret}(d) &= P_{cr}(d).\sum_{i=0}^{K}(1-P_{cr}(d))^i \\ &= P_{cr}(d)\frac{(1-P_{cr}(d))^{K+1}-1}{(1-P_{cr}(d))-1} \\ &= 1-(1-P_{cr}(d))^{K+1} \end{aligned} \tag{3.8}$$

La Figure 3.3 est un tracé de l'Equation 3.8 pour une puissance de transmission de 1 mW et différents nombres de retransmissions. On peut noter que la zone de transition n'est pas affectée par le nombre de retransmissions, mais que la zone connectée est augmentée.

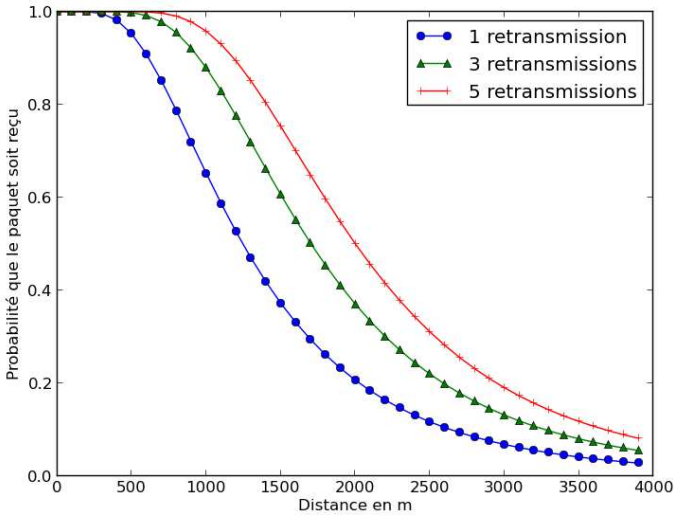


FIGURE 3.3 – Probabilité qu'un paquet soit correctement reçu en fonction de la distance émetteur-récepteur avec retransmissions

3.2.2 Topologie du réseau

Nous décrivons ici les hypothèses que nous faisons sur la topologie du réseau. Comme dans [Gupta and Kumar, 2000] et [Watteyne, 2008], N nœuds sont placés aléatoirement, suivant une distribution uniforme, sur un disque d'aire unitaire. Le puits est placé au centre du disque. Les paquets générés dans le réseau doivent rejoindre le puits, en multi-sauts. Nous supposons que les nœuds sont distribués de manière uniforme sur le disque, il y a donc en moyenne un nœud dans une surface de $\frac{1}{N}$. De cette manière on peut calculer le rayon R d'un cercle qui contient d nœuds en moyenne :

$$R = \sqrt{\frac{d}{\pi N}} \tag{3.9}$$

3.2.3 Protocoles

Nous nous intéressons particulièrement aux protocoles de routage, car nous voulons obtenir la fiabilité des transmissions de bout en bout. L'impact de la couche MAC est tout de même pris en compte.

La fiabilité au niveau MAC

Du point de vue de la fiabilité, les couches MAC peuvent être classées dans deux catégories : déterministe ou non-déterministe. La première assure que le paquet est reçu (il n'y a pas de collision ou les collisions sont résolues). Dans la seconde, des paquets peuvent être perdus à cause du protocole MAC (collisions non résolues). Dans la seconde catégorie, il y a une probabilité que le paquet ne soit pas reçu. C'est

le cas, par exemple, des protocoles de type CSMA/CA : un nœud tente d'envoyer un paquet, si une collision se produit, il va retenter après un délai aléatoire. Le nombre de tentatives étant limité, il y a donc une probabilité pour que le paquet ne soit pas reçu après l'expiration des tentatives.

La fiabilité au niveau routage

Pour le routage, nous utilisons le nombre de sauts pour atteindre le puits comme métrique. Le chemin le plus court est donc celui qui a le nombre de sauts minimum, ce paramètre est connu grâce à la construction d'un gradient : à chaque nœud est alloué un entier qui correspond au nombre de sauts qu'un paquet doit effectuer depuis ce nœud pour atteindre le puits. Dans le cas du modèle radio simplifié, le nombre de sauts pour atteindre le puits dépend de la taille de l'ensemble des voisins qui est lié à la portée radio par l'Equation 3.9. Dans le cas du modèle radio réaliste, le nombre de sauts pour atteindre le puits est difficile à évaluer. En effet, un nœud peut potentiellement communiquer avec le puits ou tout autre nœud du réseau (avec une probabilité très faible si le lien est long). On suppose donc que le gradient est construit en utilisant un seuil. Au dessus d'un seuil de probabilité de réception, deux nœuds sont considérés comme voisins. Nous supposons que les nœuds sont distribués uniformément. Les nœuds qui ont la même distance au puits en nombre de sauts forment donc des anneaux concentriques, de largeur R, centrés sur le puits. Dans le cas du modèle réaliste, la largeur des anneaux dépend du seuil de probabilité P_{th} :

$$P_{cr}(R) = P_{th} \tag{3.10}$$

P_{th} permet de sélectionner les liens considérés comme fiables. Notons qu'on peut modéliser des routages qui utilisent d'autres métriques que la fiabilité du lien pour définir les voisins : on peut, par exemple, considérer un seuil qui prend en compte l'énergie à dépenser pour réaliser un saut.

Au niveau routage, nous classons les protocoles dans deux catégories du point de vue de la fiabilité (déjà définies dans la section 2.1.2 du chapitre précédent) : le routage classique et le routage opportuniste. Avec le routage classique, un chemin est défini entre chaque source et le puits (en utilisant le gradient dans notre cas) : chaque nœud définit son prochain saut en le choisissant parmi ses voisins les plus proches du puits. Il envoie les paquets à router à ce prochain saut. Dans le cas du routage opportuniste, le paquet est diffusé aux voisins du nœud (de manière similaire à GRAB [Ye et al., 2005a], cf. section 2.1.2 du Chapitre 2). Un ensemble de relayeurs potentiels entrent en concurrence pour relayer le paquet. Les relayeurs potentiels sont les nœuds qui sont plus proches du puits que l'émetteur en terme de nombre de sauts (connu grâce au gradient). L'élection du relayeur peut être réalisée en fonction de différentes métriques (force du signal, niveau d'énergie, aléatoire, ...). Dans ce chapitre, nous considérons qu'un unique relayeur est choisi arbitrairement parmi les relayeurs potentiels qui ont reçu le paquet. Nous ne considérons donc pas les réplications de paquets (dues au mécanisme de sélection du relayeur comme évoqué dans le chapitre précédent) dans ce chapitre. En réalité, ces réplications, d'une part, augmentent la

fiabilité (redondance des paquets) et d'autre part, augmentent la congestion et les collisions dans le réseau. On montre dans le Chapitre 5, que malgré ces risques, le protocole opportuniste que nous proposons a de meilleures performances (énergie, taux de livraison) que des solutions classiques.

Les approches de routage opportuniste et classique se comportent de manières très différentes vis-à-vis de la fiabilité des transmissions de bout en bout comme nous le montrons dans l'étude théorique de la section suivante.

3.3 Etude de la fiabilité des RCsF

Dans cette section, nous exprimons la probabilité de succès d'une transmission de bout en bout grâce au modèle décrit précédemment. Nous donnons l'expression de cette probabilité pour le routage classique et opportuniste en utilisant les modèles de lien radio simplifié et réaliste.

3.3.1 Lien radio simplifié

Routage classique

Nous considérons que le mécanisme de routage sélectionne un chemin depuis la source jusqu'au puits. Un nœud appartenant au chemin relaye le paquet au voisin qui est le suivant dans le chemin. La probabilité qu'un saut réussisse, c'est-à-dire que le paquet soit reçu, est P_{bcr}. Notons, que dans cette étude, nous exprimons les probabilités de transmissions de bout en bout sans prendre en compte les retransmissions. Pour les prendre en compte, il suffit de substituer P_{bcr} par $1-(1-P_{cr}(d))^{K+1}$ déduit de l'Equation 3.3. La probabilité que le paquet soit correctement reçu par le puits après H sauts est donnée par l'Equation 3.11. Dans la suite de ce chapitre, H dénote l'anneau de gradient d'origine de la transmission (qui correspond au nombre de sauts pour atteindre le puits) et h est un indice sur les anneaux de gradient plus proches du puits que H.

$$P_{e2e\_1u}(H) = \prod_{h=1}^{H} P_{bcr} = (P_{bcr})^H \tag{3.11}$$

Pour un nœud donné, le nombre de sauts pour atteindre le puits dépend de la portée du nœud (R, qui correspond à la largeur d'un anneau de gradient) :

$$H = \left\lceil \frac{1}{R\sqrt{\pi}} \right\rceil \tag{3.12}$$

$\frac{1}{\sqrt{\pi}}$ est le rayon du disque d'aire unitaire. La probabilité de succès d'une communication de bout en bour P_{e2e_1u} tend vers zéro quand R tend vers zéro (car H tend vers $+\infty$ et $0 \leq P_{bc} \leq 1$) et est égale à P_{bcr} quand $R \geq \frac{1}{\sqrt{\pi}}$. Cela signifie, dans le cas du modèle de lien simplifié, que lorsque il y a moins de sauts, les transmissions sont plus fiables : c'est dû au fait que P_{bcr} ne décroit pas en fonction de la distance

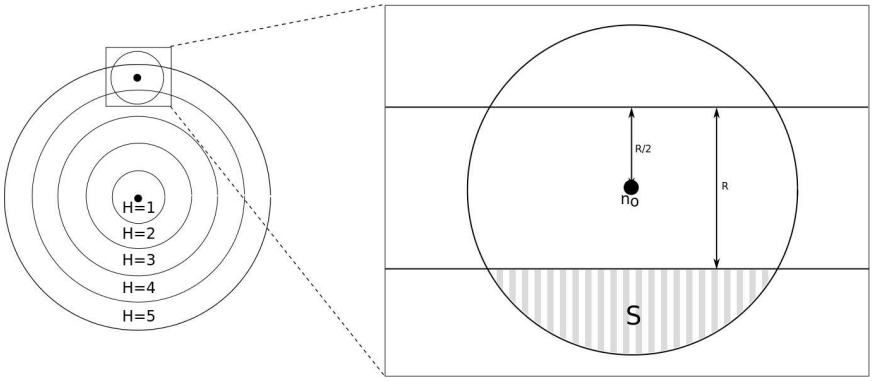


FIGURE 3.4 – Approximation d'un anneau de gradient en droites parallèles

émetteur-récepteur. Le modèle de lien simplifié ne permet donc pas de représenter le fait que les transmissions longues portées sont moins fiables.

On remarque qu'en réalité, utiliser une valeur de R importante n'est pas seulement problématique à cause de la non-fiabilité des transmissions longues portées (on peut les fiabiliser en augmentant la puissance de transmission), mais aussi parce que cela réduit les possibilités de réutilisation spatiale (nombre de transmissions simultanées dans le réseau) et donc cela réduit la capacité du réseau comme mentionné dans [Gupta and Kumar, 2000].

Routage opportuniste

Dans le cas du routage opportuniste, un ensemble de relayeurs potentiels reçoivent le paquet. Cet ensemble est défini par le protocole de routage utilisé. Dans notre cas, ce sont les nœuds plus proches du puits que l'émetteur en terme de nombre de sauts. Grâce au gradient, nous pouvons déterminer le pourcentage de voisins d'un nœud qui ont un nombre de sauts inférieur pour atteindre le puits. Si le gradient n'est pas défini, ce pourcentage doit être déterminé en fonction du protocole de routage utilisé.

Comme représenté sur la Figure 3.4, nous supposons que si un anneau de gradient h a un rayon suffisamment grand, il peut être considéré comme deux droites parallèles à l'échelle d'un nœud. Cette approximation est moins bonne pour les anneaux proches du puits : une correction peut donc être apportée (le calcul exact sera donné dans le Chapitre 4). Avec cette approximation, et étant donné que les nœuds sont répartis uniformément sur le disque d'aire unitaire, un nœud est placé en moyenne à égale distance des limites de l'anneau. Cela signifie, qu'en moyenne, sa portée radio recouvre une surface S (cf. Figure 3.4) de l'anneau précédent (à savoir l'anneau adjacent plus proche du puits). La surface S est la surface d'un segment de cercle :

$$S = \left(\frac{\pi}{3} - \frac{\sqrt{3}}{4}\right) R^2 \tag{3.13}$$

Le rapport entre la surface S et la surface définie par le rayon de transmission est donnée par l'Equation 3.14.

$$r = \frac{S}{\pi R^2} = \left(\frac{1}{3} - \frac{\sqrt{3}}{4\pi}\right) \tag{3.14}$$

Nous savons que le degré des nœuds est égale à d (hypothèse du modèle de lien simplifié). Comme les nœuds sont distribués de manière uniforme, le nombre de nœuds dans une aire donnée est proportionnel à la surface de cette aire. Nous définissons donc m le nombre de relayeurs potentiels comme étant : $m = rd$.

La probabilité de succès d'un saut, dans le cas du routage opportuniste, est la probabilité qu'au moins un relayeur potentiel reçoive le paquet. La probabilité que la transmission de bout en bout soit un succès est la probabilité que tous les sauts réussissent. Elle est donnée par l'Equation 3.15 pour un chemin de H sauts.

$$\begin{aligned} P_{e2e\_1b}(H) &= P_{bcr} \prod_{h=1}^{H-1} (1 - \prod_{j=1}^{m} (1 - P_{bcr})) \\ &= P_{bcr}(1 - (1 - P_{bcr})^m)^{H-1} \end{aligned} \tag{3.15}$$

Pour le dernier saut, le seul récepteur est le puits, h varie donc entre 1 et $H - 1$ dans la somme. Cela implique aussi que le dernier saut est moins fiable que les autres, car si le puits ne reçoit pas le paquet, il sera perdu exactement comme dans le cas du routage classique. On peut donc voir le puits comme un goulot d'étranglement de la fiabilité du réseau : le dernier saut est le moins fiable et limite donc la fiabilité des transmissions de bout en bout. On peut aussi noter que le nombre de relayeurs potentiels m devrait diminuer pour les anneaux proches du puits, car ils sont plus petits. Pour prendre en compte cet effet, la surface S doit être calculée en utilisant la technique présentée dans le Chapitre 4. Dans ce cas m devient une fonction de h.

Quand m tend vers $+\infty$, $(1 - P_{bcr})^m$ tends vers 0 donc la probabilité de succès d'une transmission de bout en bout P_{e2e_1b} tend vers P_{bcr}. Cela signifie que plus il y a de relayeurs potentiels, plus les transmissions de bout en bout sont fiables. Cependant, le puits est un goulot d'étranglement pour la fiabilité, ce qui fait que P_{e2e_1b} ne tend pas vers 1 quand m tend vers $+\infty$. Il est possible d'élargir ce goulot en ajoutant des puits (cet ajout n'a pour but que d'augmenter la fiabilité du dernier saut, cela n'augmente pas la capacité : les puits sont reliés à une station de base et agissent comme un seul puits). Dans ce cas l'Equation 3.15 devient :

$$P_{e2e\_1b}(H) = (1 - (1 - P_{bcr})^{m_s}).(1 - (1 - P_{bcr})^m)^{H-1} \tag{3.16}$$

avec m_s le nombre de puits. Cela permet d'augmenter la fiabilité à moindre coût énergétique. En effet, augmenter le nombre de puits ne réduit pas la durée de vie du réseau. D'autre part, on retrouve les mêmes problèmes que dans le cas du routage classique sur la fiabilité des transmissions longues portées et la réutilisation spatiale.

Comparaison entre routage classique et opportuniste

Théorème 3.3.1 *Le routage opportuniste est au moins aussi fiable que le routage classique :* $P_{e2e\_1b} \geq P_{e2e\_1u}$

Preuve

$$\begin{aligned}
1 - P_{bcr} &\geq (1 - P_{bcr})^m \\
1 - (1 - P_{bcr}) &\leq 1 - (1 - P_{bcr})^m \text{ car } 1 - P_{bcr} \text{ et } (1 - P_{bcr})^m \leq 1 \\
P_{bcr} &\leq 1 - (1 - P_{bcr})^m \\
(P_{bcr})^H &\leq P_{bcr}(1 - (1 - P_{bcr})^m)^{H-1} \\
P_{e2e_1u} &\leq P_{e2e_1b} \quad \blacksquare
\end{aligned}$$

Le routage opportuniste est plus fiable que le classique mais requiert une plus grande consommation d'énergie. Intuitivement, la solution opportuniste propose une plus grande diversité de liens à chaque saut. Cependant, elle met en jeu plus de nœuds, les relayeurs potentiels doivent être éveillés pour recevoir le paquet de l'émetteur. Un mécanisme pour élire le relayeur effectif doit aussi être implémenté et ce mécanisme mène à des réplications de paquets [Biswas and Morris, 2005] [Lampin et al., 2012] [Ye et al., 2005a]. Ces mécanismes consomment de l'énergie, il y a donc un compromis entre énergie et fiabilité que nous mettrons en évidence dans le Chapitre 5 lors de l'évaluation et la comparaison des performances de RTXP et d'une solution d'acheminement de données qui utilise le schéma de routage classique.

3.3.2 Lien radio réaliste

Dans cette section, nous utilisons le modèle de propagation *log-normal shadowing*, plus réaliste que le modèle de lien simplifié, pour exprimer la fiabilité des transmissions de bout en bout. La probabilité qu'un paquet soit correctement reçu dépend, dans ce cas, de la distance émetteur-récepteur : pour déterminer la probabilité de succès d'un saut il faut donc estimer cette distance.

Routage classique

Dans le cas du routage classique, un chemin depuis chaque source jusqu'au puits est prédéfini. Cependant dans le cas du modèle de lien réaliste, tous les nœuds du réseau peuvent potentiellement communiquer avec tous les autres (même si dans certains cas cette probabilité est très faible). Deux cas sont donc possibles quand un paquet est envoyé : soit le relayeur est le nœud qui est le suivant dans le chemin, soit cela peut être n'importe quel nœud appartenant chemin et étant plus proche du puits, comme dans [Lampin et al., 2012]. On nomme ce deuxième cas routage hybride.

Dans le cas classique, un paquet provenant d'un nœud de l'anneau H du gradient doit parcourir une distance de $H \times R$ car il réalise H sauts. R est dans ce cas la largeur d'un anneau qui est fonction de la construction du gradient. En effet on

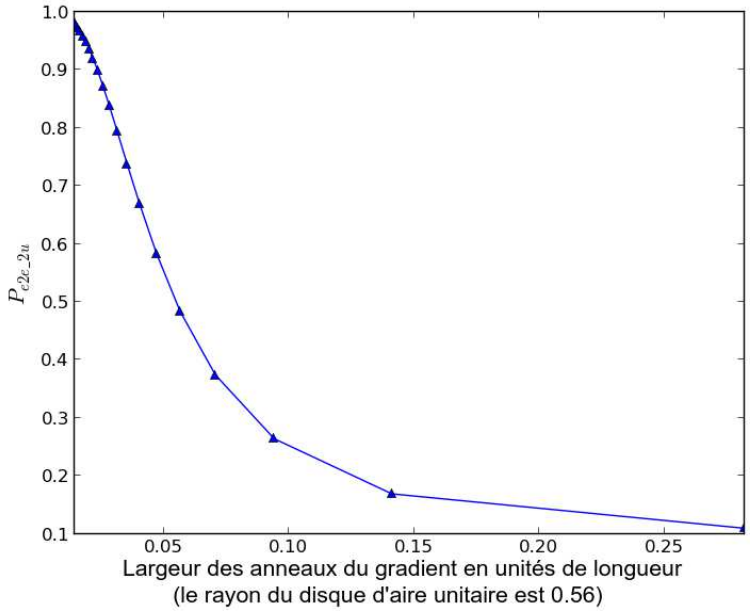


FIGURE 3.5 – Influence de la construction du gradient sur la fiabilité

rappelle que $P_{cr}(R) = P_{th}$ avec P_{th} la probabilité seuil durant la construction du gradient. La probabilité de succès d'une transmission de bout en bout est obtenue en substituant P_{bcr} par $P_{cr}(R)$ dans l'Equation 3.11. La Figure 3.5 représente la fiabilité des transmissions de bout en bout d'un nœud pour différents gradients : la position du nœud ne change pas, mais on fait varier la largeur des anneaux du gradient en changeant la valeur de P_{th}. Quand la largeur d'un anneau est grande, il faut peu de sauts pour atteindre le puits, mais les sauts sont peu fiables. Quand la largeur des anneaux est faible, il faut plus de sauts pour atteindre le puits, mais la transmission de bout en bout est plus fiable. Un nombre plus important de sauts signifie un délai plus important du fait des délais d'accès au canal, de transmission et de propagation. Ce résultat est utilisé par [Chipara et al., 2006] avec le protocole RPAR (présenté dans la section 2.2.2 du Chapitre 2) qui fait un compromis entre fiabilité et délai.

Routage hybride

Dans le cas hybride, les nœuds plus proches du puits que l'émetteur du paquet et appartenant au chemin de routage, peuvent le relayer. Nous supposons qu'un nœud relaye un paquet qu'il vient de recevoir, seulement si aucun nœud plus proche du puits ne l'a reçu (seuls les nœuds appartenant au chemin de routage peuvent recevoir le paquet). Ce mécanisme est implémenté dans [Lampin et al., 2012]. Pour un paquet provenant de l'anneau de gradient H, il y a $H - 1$ relayeurs potentiels. Pour chaque nœud du chemin, la probabilité d'être élu est égale à la probabilité d'avoir reçu le paquet et qu'aucun nœud plus proche du puits ne l'ait reçu. La probabilité du succès de la transmission de bout en bout est la somme des probabilités que chaque nœud

du chemin soit un relayeur, et la probabilité que le paquet atteigne le puits depuis le relayeur. La probabilité d'atteindre le puits depuis le relayeur est définie de la même façon que probabilité de bout en bout : c'est donc une fonction récursive définie par les Equations 3.17 et 3.18.

$$P_{e2e\_2u}(0) = 1 \tag{3.17}$$

$$P_{e2e\_2u}(H) = \sum_{h=1}^{H-1} \left\{ \prod_{j=0}^{h-1} [1 - P_{cr}((H-j)R)] \times P_{cr}((H-h)R) \times P_{e2e\_2u}(h) \right\} \tag{3.18}$$

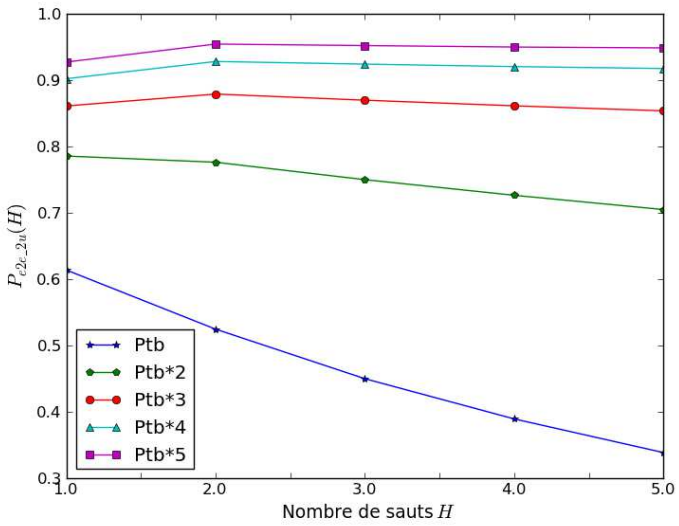


FIGURE 3.6 – Probabilité du succès d'une transmission de bout en bout dans le cas hybride (P_{e2e_2u}) en fonction de H pour différentes puissances de transmission

La Figure 3.6 représente l'Equation 3.18 avec H variant de 1 à 5 et différentes puissances de transmission (on prend une puissance de base P_{tb} que l'on multiplie). Faire varier la puissance de transmission est équivalent à faire varier le seuil de probabilité P_{th} : pour une largeur d'anneau R donnée, les Equations 3.10 et 3.5 (respectivement p.47 et p.44) impliquent qu'une augmentation de la puissance induit une augmentation du seuil P_{th}. Faire varier H signifie que l'on place la source du paquet de plus en plus loin du puits. On observe que pour les puissances de transmission les plus faibles (P_{tb} et $2P_{tb}$), la probabilité décroît strictement quand H augmente. Ce n'est pas le cas pour $3P_{tb}$, $4P_{tb}$ et $5P_{tb}$. Dans ces cas, la probabilité est plus grande pour $H = 2$ que pour $H = 1$: c'est dû au fait que la probabilité de succès de la transmission directe depuis l'anneau $H = 2$ au puits est assez grande pour compenser la différence entre la probabilité de succès d'un saut et celle de deux sauts. C'est le cas par exemple, si la probabilité de succès d'un saut est de 0.9 et la probabilité de succès

de la transmission directe depuis l'anneau $H = 2$ est de 0.15 : on a $P_{e2e\_2u}(1) = 0.9$ et $P_{e2e\_2u}(2) = 0.9 * 0.9 + 0.15 = 0.96$.

La fiabilité dans le cas hybride est plus grande que dans le cas du routage classique. Intuitivement, on constate qu'il y a plus de possibilités pour atteindre le puits. Cependant, cette solution requiert que les nœuds du chemin soient éveillés à chaque saut : cela augmente la consommation d'énergie.

Routage opportuniste

Avec le routage opportuniste, un nœud relaye un paquet s'il est plus proche du puits que l'émetteur et qu'aucun nœud plus proche du puits n'a reçu le paquet. Contrairement au cas hybride, il y a plusieurs relayeurs potentiels dans chaque anneau h (on rappelle que $h < H$). Pour calculer la fiabilité, il faut donc déterminer :

- le nombre de relayeurs potentiels dans chaque anneau ;
- le nombre de nœuds plus proches du puits qu'un relayeur potentiel ;
- la distance entre ces nœuds et l'émetteur (car la probabilité de réception dépend de la distance émetteur-récepteur).

Dans la suite de cette section, nous établissons les expressions de ces paramètres et nous les intégrons dans l'expression de la fiabilité de bout en bout.

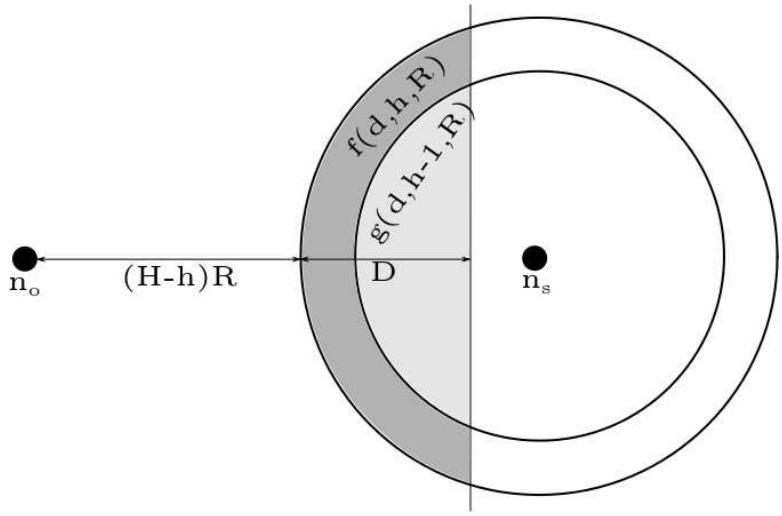


FIGURE 3.7 – Relation entre les surfaces de segments d'anneaux du gradient et la distance émetteur-récepteur

Le nombre de nœuds dans l'anneau h noté m_h et le nombre de nœuds plus près du puits noté m_{h-} sont donnés respectivement par les Equations 3.19 et 3.20.

$$m_h = \lfloor (\pi(Rh)^2 - \pi[R(h-1)]^2)N \rfloor \tag{3.19}$$

$$m_{h-} = \lfloor (\pi[R(h-1)]^2)N \rfloor \tag{3.20}$$

Pour calculer les probabilités de réception des paquets à l'aide de l'Equation 3.5, il faut évaluer les distances entre l'émetteur et les récepteurs potentiels. Comme nous

supposons que les nœuds sont répartis uniformément sur le disque d'aire unitaire, le nombre de nœuds dans une surface donnée est proportionnel à l'aire de cette surface. Dans notre cas nous définissons des aires comme représenté sur la Figure 3.7. Les aires de segments de cercle f et g sont définies en fonction de la distance D, donc la distance émetteur-récepteur est liée à la surface et la surface est liée au nombre de nœuds. On peut donc déterminer la distance entre l'émetteur et le $n^{ème}$ nœud de l'anneau h.

La fonction f est la surface du segment de l'anneau h, et est définie comme suit :

– si $0 < D < R$:

$$f(D,h,R) = \cos^{-1}\left(1-\frac{D}{hR}\right)(hR)^2 - (Rh-D)\sqrt{2hRD-D^2} \tag{3.21}$$

– si $R \leq D \leq (2h-1)R$:

$$\begin{aligned} f(D,h,R) &= \cos^{-1}\left(1-\frac{D}{hR}\right)(hR)^2 - (Rh-D)\sqrt{2hRD-D^2} \\ &\quad -\cos^{-1}\left(1-\frac{D-R}{(h-1)R}\right)((h-1)R)^2 \\ &\quad +(R(h-1)-D+R)\sqrt{2(h-1)R(D-R)-(D-R)^2} \end{aligned} \tag{3.22}$$

– si $D \geq (2h-1)R$:

$$\begin{aligned} f(D,h,R) &= \cos^{-1}\left(1-\frac{D}{hR}\right)(hR)^2 - (Rh-D)\sqrt{2hRD-D^2} \\ &\quad -\cos^{-1}\left(1-\frac{D-R}{(h-1)R}\right)((h-1)R)^2 \end{aligned} \tag{3.23}$$

La fonction inverse par rapport à D donne la distance en fonction de la surface (avec $S = n/N$ pour le $n^{ème}$ voisin) :

$$f^{-1}(S,h,R) = D \tag{3.24}$$

De même, la surface pour les nœuds plus proches du puits est donnée par l'Equation 3.25 :

$$g(D,h,R) = \cos^{-1}\left(1-\frac{D}{hR}\right)(hR)^2 - (Rh-D)\sqrt{2hRD-D^2} \tag{3.25}$$

et $g^{-1}(S,h,R)$ est définie de la même manière que f^{-1}.

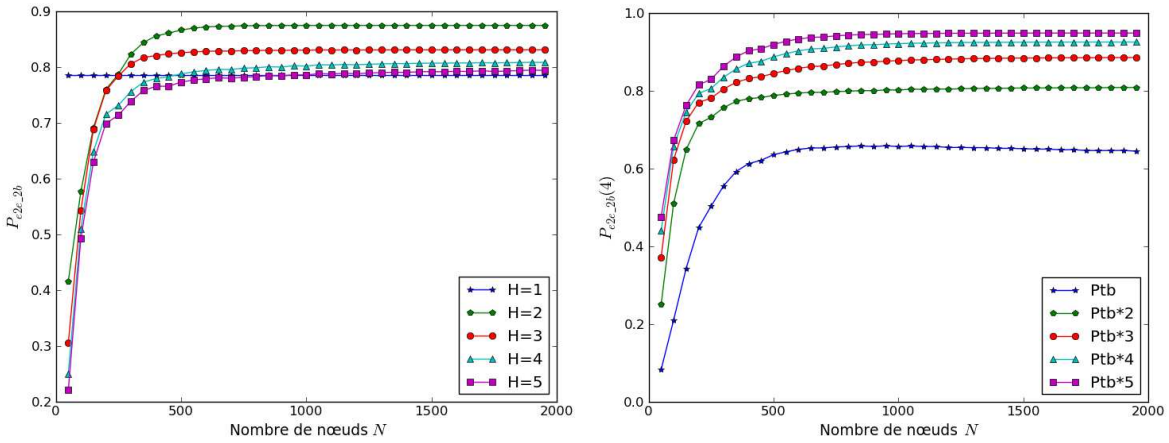
Comme dans le second cas du routage classique, la probabilité de succès d'une transmission de bout en bout est une fonction récursive définie par les Equations 3.26 et 3.27.

$$P_{e2e\_2b}(0) = 1 \tag{3.26}$$

$$P_{e2e\_2b}(H) = \sum_{h=0}^{H-1}\left\{\left[1-\prod_{j=1}^{m_h}(1-P_{cr}(f^{-1}(\frac{j}{N},h,R)+R\times(H-h)))\right]\right.$$
$$\times\left[\prod_{k=1}^{m_{h-}}(1-P_{cr}(g^{-1}(\frac{k}{N},h-1,R)+R\times(H-(h-1))))\right]$$
$$\left.\times P_{e2e\_2b}(h)\right\} \quad (3.27)$$

La probabilité qu'une transmission de bout en bout, provenant d'un nœud de l'anneau H, réussisse est la somme des probabilités, pour chaque anneau h ($h \in [1, H-1]$), qu'au moins un nœud de l'anneau h reçoive le paquet et qu'aucun nœud plus proche du puits ne le reçoive et la probabilité d'atteindre le puits depuis l'anneau h.

On peut noter que lorsque $H = 0$, f^{-1} et g^{-1} sont égales à 0, m_h est égal à 1 (le puits) et m_{h-} est égal à 0 (de même, pour $H = 1$, on a m_{h-} égal à 1 et g^{-1} égal à 0).



(a) H varie de 1 à 5 et la puissance de transmission est $P_{tb} \times 2$

(b) La puissance de transmission varie de P_{tb} à $P_{tb} \times 5$ et $H = 4$

FIGURE 3.8 – Probabilité du succès d'une transmission de bout en bout dans le cas opportuniste (P_{e2e_2b}) en fonction du nombre de nœuds N

La Figure 3.8(a) représente l'Equation 3.27 avec le nombre de nœuds N variant de 50 à 2000 et H de 1 à 5 avec une puissance de transmission de $P_{tb} \times 2$. La probabilité de succès d'une transmission de bout en bout augmente avec N jusqu'à une valeur maximale qui dépend de H. Cette valeur maximale est inférieure à 1. En augmentant le nombre de nœuds N, le nombre de relayeurs potentiels dans chaque anneau croît. Cela rend les transmissions plus fiables car cela fait augmenter la diversité. En revanche, il n'y a toujours qu'un seul puits donc la fiabilité du dernier saut n'augmente pas. C'est pour cela que la fiabilité converge vers une valeur inférieure à 1. Comme dans le cas du modèle de lien simplifié, le puits peut être vu comme un goulot d'étranglement de la fiabilité du réseau.

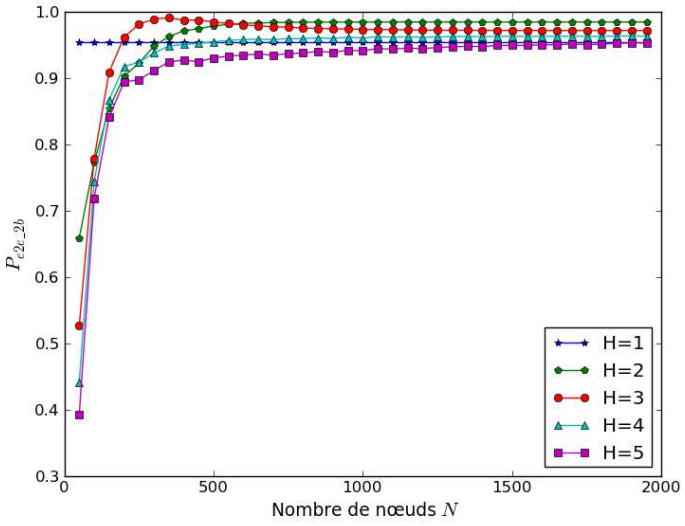


FIGURE 3.9 – Probabilité du succès d'une transmission de bout en bout dans le cas opportuniste (P_{e2e_2b}) en fonction du nombre de nœuds N pour différents H avec 2 puits pour $P_{tb} \times 2$

La Figure 3.9 représente l'Equation 3.27 pour N variant de 50 à 2000 et H de 1 à 5 avec une puissance de transmission de $P_{tb} \times 2$ et 2 puits. On observe que la fiabilité augmente d'environ 0.15 par rapport au cas où il n'y a qu'un puits, représenté par la figure 3.8(a).

La Figure 3.8(b) est une représentation de l'Equation 3.27 avec N variant de 50 à 2000 et la puissance de transmission variant de $1 \times P_{tb}$ à $5 \times P_{tb}$ pour $H = 4$. Dans ce cas aussi, la fiabilité atteint une valeur maximale. Cette valeur augmente avec la puissance de transmission, car les transmissions sont plus fiables. On peut remarquer qu'en ajoutant des puits, on augmente la fiabilité sans augmenter la puissance de transmission, donc sans réduire la durée de vie du réseau (bien sûr, les nouveaux puits consomment de l'énergie).

3.3.3 Impact de la couche MAC

Dans les sections précédentes nous considérons que les pertes de paquets ne sont pas dues au protocole MAC, mais seulement au canal de propagation radio non-fiable. Nous considérons maintenant que la probabilité de recevoir un paquet est la probabilité que le paquet ne soit perdu ni à cause de la couche MAC ni à cause du lien radio non-fiable. Les pertes dues à la couche MAC proviennent des collisions non résolues comme évoqué dans le Chapitre 2. On se place ici dans le cas du modèle radio simplifié, mais les conclusions sont les mêmes pour le modèle réaliste, car la probabilité due à la couche MAC s'applique de la même façon.

Dans le cas du routage classique, seule la probabilité de réception à chaque saut change. Dans l'Equation 3.11, P_{bcr} est remplacé par $P_{bcr\_nodet} = P_{bcr} \times P_{no\_coll}$ avec

P_{no_coll} la probabilité qu'il n'y ait pas de perte de paquet due à une collision au niveau du récepteur. L'Equation 3.11 devient alors :

$$P_{e2e\_1u}(H) = (P_{bcr} \times P_{no\_coll})^H \tag{3.28}$$

Dans le cas du routage opportuniste, s'il y a une perte de paquet due à une collision, le paquet a des chances d'être non décodable pour la plupart des relayeurs potentiels. Dans le pire des cas, aucun relayeur potentiel ne reçoit le paquet. Dans ce cas, l'Equation 3.15 devient :

$$P_{e2e\_1b} = (P_{bcr} \times P_{no\_coll}) \times ((1 - (1 - P_{bcr})^m) \times P_{no\_coll})^{H-1} \tag{3.29}$$

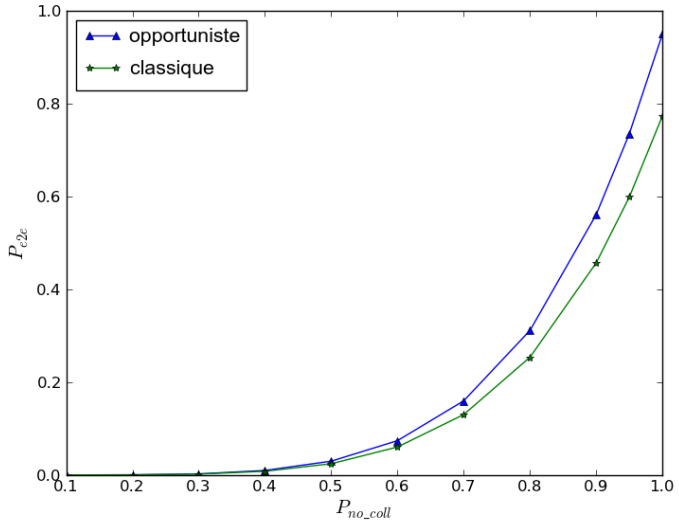


FIGURE 3.10 – P_{e2e} en fonction de P_{no_coll} pour $H = 5$ et $P_{bcr} = 0.95$

La Figure 3.10 représente l'impact de la probabilité de ne pas avoir de collision P_{no_coll} sur la fiabilité pour $H = 5$ et $P_{bcr} = 0.95$. On constate que pour des valeurs faibles de P_{no_coll}, les performances des solutions classiques et opportunistes sont similaires. C'est dû au fait que l'impact des collisions sur le mécanisme opportuniste est important car la collision détruit la diversité crée par ce mécanisme.

3.4 Comparaison avec des simulations

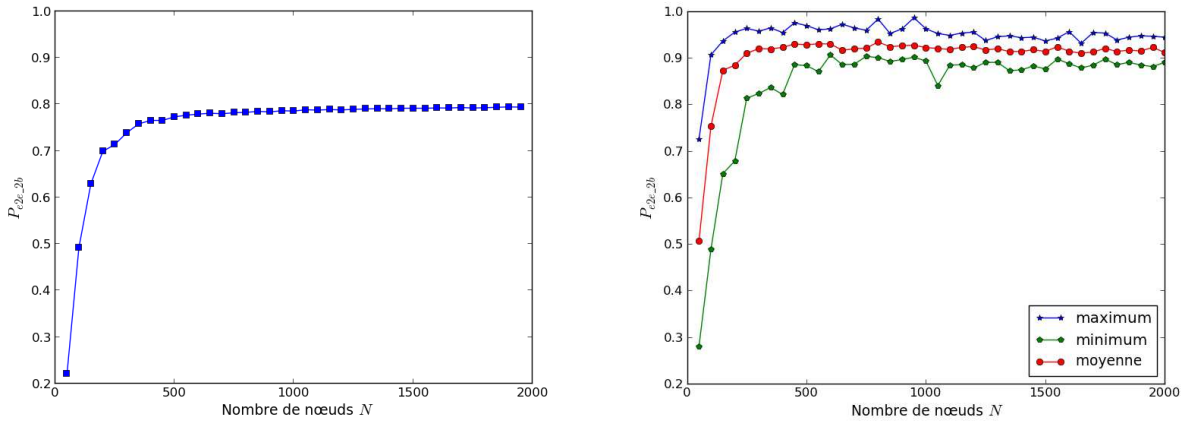
Dans cette section, nous comparons les résultats théoriques avec des résultats de simulation pour valider notre modèle théorique. Pour effectuer les simulations, nous avons développé un simulateur * en langage Python. Nous avons choisi de développer cette solution légère plutôt que d'utiliser plate-forme de simulation classique,

∗. Le code du simulateur est visible à cette adresse : `http://perso.citi.insa-lyon.fr/amouradia/code/reliability/`

car nous ne voulions pas entrer dans les détails des protocoles : nous ne modélisons pas explicitement le temps (les transmissions sont instantanées), les données et les communications concurrentes (un seul paquet se déplace dans le réseau). Cette solution légère permet d'augmenter la vitesse de simulation et donc d'en réaliser de nombreuses.

Durant les simulations, un paquet est routé vers le puits depuis son nœud d'origine, la simulation se termine soit quand le paquet est reçu par le puits, soit quand il est perdu. La perte des paquets due au *shadowing* est simulée de la manière suivante : le PER au niveau des récepteurs est calculé en fonction de la distance émetteur-récepteur suivant l'Equation 3.7. Chaque récepteur tire aléatoirement, en suivant une loi uniforme, un réel entre 0 et 1, si le réel est plus grand que le PER, le paquet est reçu, sinon le paquet est perdu.

Les nœuds sont placés de façon aléatoire et uniforme sur un disque d'aire unitaire. On fait varier le nombre de nœuds de 50 à 2000 par pas de 50. Pour chaque point, on effectue 20 simulations et dans chaque simulation, 1000 paquets sont émis. L'émetteur d'un paquet est choisi de manière aléatoire dans l'anneau de gradient considéré.



(a) Résultats théoriques

(b) Résultats de simulations

FIGURE 3.11 – Probabilité de succès d'une transmission de bout en bout (P_{e2e_2b}) en fonction de N pour $H = 5$ et une puissance de transmission de $P_{tb} \times 2$

Les Figures 3.11(a) et 3.11(b) permettent de comparer les résultats théoriques et de simulation pour du routage opportuniste, avec $H = 5$ et $Pt = P_{tb} \times 2$. On observe que les courbes évoluent de la même manière en fonction du nombre de nœuds N. On retrouve le même phénomène de goulot d'étranglement de fiabilité dans les simulations déjà constaté pour les résultats théoriques. On constate cependant, que dans les résultats de simulations, la fiabilité observée est un peu plus importante que celle prévue par le modèle théorique. C'est dû au fait que dans le modèle théorique, la fiabilité du dernier saut est donnée par $P_{cr}(R)$, or en réalité la distance du dernier saut est inférieur ou égale à R. Nous prenons donc en compte le pire cas dans le modèle théorique, c'est pour cela que notre modèle est un peu plus pessimiste que les simulations.

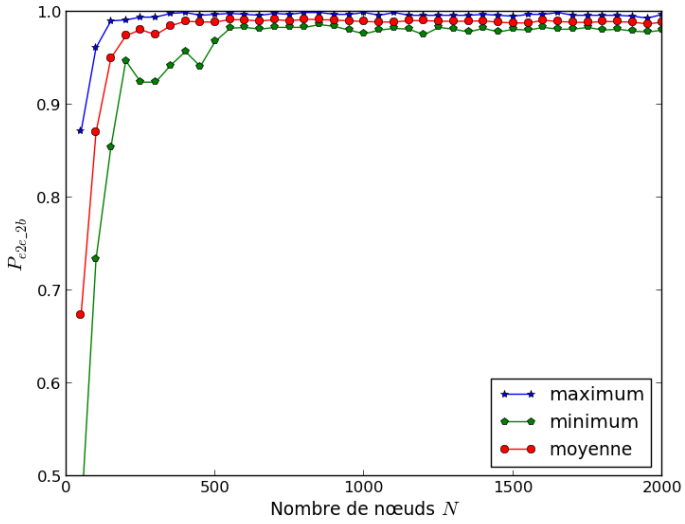


FIGURE 3.12 – Résultats de simulations : P_{e2e_2b} en fonction de N pour $H = 5$ et 2 puits

La Figure 3.12 représente les résultats de simulations pour des paramètres similaires à ceux de la Figure 3.11(b), mais avec 2 puits. On observe que dans le cas de la simulation, l'ajout d'un puits augmente également la fiabilité. En comparant ces résultats à ceux de la Figure 3.9, on constate que les résultats théoriques sont, dans ce cas également, plus pessimistes que les simulations, dû aux raisons précédemment évoquées.

Les résultats de simulations montrent tout de même que notre modèle capture bien la fiabilité des topologies aléatoires.

3.5 Conclusion

Ce chapitre présente une étude théorique qui permet d'évaluer la fiabilité qu'il est possible d'obtenir avec un RCsF. La fiabilité étant définie comme la probabilité du succès d'une transmission de bout en bout. Nous nous intéressons notamment au cas où la probabilité du succès d'un saut dépend de la distance entre l'émetteur et le récepteur, ce qui n'est pas, à notre connaissance, étudié dans la littérature. Grâce à l'hypothèse d'uniformité de la distribution des nœuds, nous parvenons à donner une fiabilité non pas pour des instances arbitraires de réseaux, mais pour toute topologie générée aléatoirement de manière uniforme. Nous nous intéressons à la différence de fiabilité des schémas de routage classique et opportuniste. Nous déduisons des résultats que le schéma opportuniste est plus fiable que le classique, mais au prix d'une consommation énergétique plus importante. Il est étonnant qu'aucun protocole temps-réel de la littérature n'utilise ce schéma de routage alors même que les protocoles temps-réel sont conçus pour servir des applications critiques qui nécessitent un niveau de fiabilité élevé. Nous mettons aussi en évidence le problème du goulot

d'étranglement de fiabilité au niveau du puits pour le mécanisme opportuniste. L'utilisation de plusieurs puits permet alors d'augmenter la fiabilité sans réduire la durée de vie du réseau.

D'autres modèles de propagation peuvent être utilisés avec le modèle de fiabilité que nous proposons. Notamment, une limite de notre étude vient du fait que le modèle de propagation que nous utilisons ne prend pas en compte la corrélation spatiale dans la probabilité de réception. En effet, les liens sont considérés comme indépendants, alors qu'en réalité, il y a une forte probabilité pour que deux nœuds proches soient soumis aux mêmes conditions de canal.

Dans la suite de ce document, nous proposons une solution d'acheminement de données en temps-réel qui tire parti des conclusions de l'étude de fiabilité de ce chapitre (utilisation du routage opportuniste et évaluations de performances avec plusieurs puits). Dans le chapitre suivant, nous nous intéressons à l'organisation logique du réseau dans le but de permettre le routage en bornant le nombre de sauts pour atteindre le puits et de permettre des accès déterministes (en temps borné) au canal. Nous développons pour cela un système de coordonnées virtuelles à une dimension.

Chapitre 4

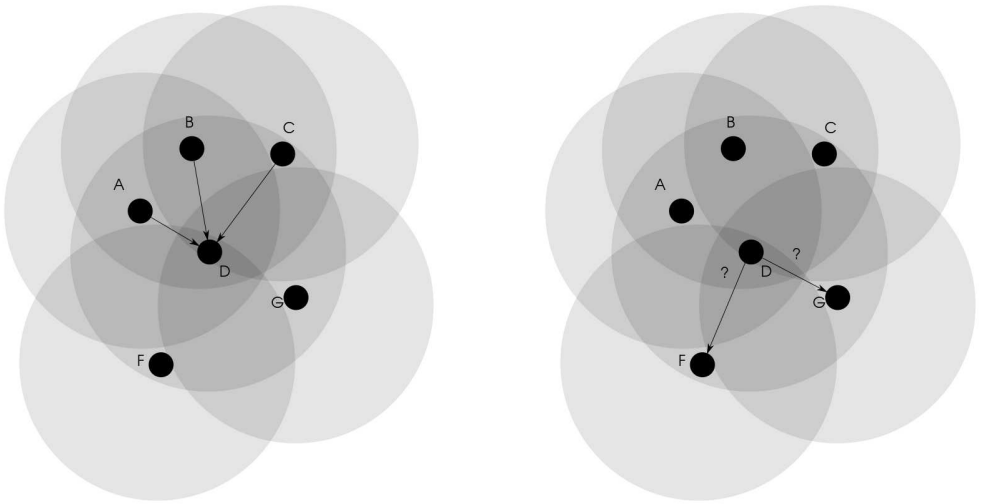
Organiser le réseau pour introduire du déterminisme

Les protocoles temps-réel dur doivent fournir des garanties strictes. Pour pouvoir fournir ces garanties, le comportement de ces protocoles doit être prédictible et donc déterministe. Au niveau MAC, afin de minimiser le nombre de collisions, il faut pouvoir discriminer les nœuds dans un domaine d'interférence. Au niveau routage, il faut pouvoir garantir que la longueur de la route est bornée et qu'à chaque saut un unique relayeur est élu. Dans ce cas aussi, il faut être capable de discriminer les voisins d'un nœud. Dans ce chapitre et ainsi que dans le reste de la première partie de ce document, nous supposons un modèle d'interférences à deux sauts, c'est-à-dire que les nœuds interférants (définition 2.1.1) d'un nœud sont ceux qui se trouve dans son 2-voisinage (définition 2.1.3). Les nœuds qui sont dans un même voisinage sont donc en compétition pour accéder au médium. Par exemple, sur la Figure 4.1(a), A, B et C ont un paquet à envoyer à D au même instant : il faut donc pouvoir les discriminer pour organiser les accès au canal et ainsi éviter les collisions. Au niveau routage, il faut pouvoir choisir un relayeur. Sur la Figure 4.1(b), le nœud D veut router un paquet. Si deux de ses voisins ont le même identifiant, cela mène à une duplication de paquet qui induit une augmentation de la concurrence pour accéder au médium ainsi qu'une consommation d'énergie supplémentaire.

Pour permettre de discriminer les nœuds et assurer un routage borné, nous proposons un Système de Coordonnées Virtuelles (SCV) à une dimension qui devra avoir les propriétés suivantes :

- la coordonnée est unique dans un 2-voisinage (pour discriminer les nœuds) ;
- la coordonnée permet de déterminer le nombre de sauts pour atteindre le puits (pour pouvoir le borner) ;
- la coordonnée donne une information sur la connectivité du nœud (pour sélectionner le relayeur).

Dans ce chapitre, nous détaillons le SCV proposé : il est basé sur un gradient qui est affiné pour pouvoir discriminer les nœuds. Nous montrons que le premier point n'est pas respecté : il existe une probabilité non nulle que des nœuds d'un même 2-voisinage obtiennent la même coordonnée. Cependant, nous évaluons, grâce à un modèle théorique et par simulation, le nombre moyen de coordonnées non-uniques



(a) A, B et C ont un paquet à envoyer à D au même instant : il faut donc pouvoir les discriminer pour organiser les accès au canal et ainsi éviter les collisions.

(b) D veut router un paquet dans la direction du puits, il doit pouvoir différencier ses voisins pour leur adresser le paquet.

FIGURE 4.1 – Nécessité d'utiliser un identifiant unique aux niveaux MAC et routage

dans un 2-voisinage et montrons qu'il est non seulement faible, mais aussi qu'il reste constant quand le degré du réseau augmente.

Comme évoqué dans les chapitres précédents, la topologie du réseau évolue du fait de plusieurs phénomènes : les liens non-fiables peuvent disparaître ou apparaître, les nœuds épuisent leur batterie ou bien des nœuds sont ajoutés au réseau. Cela modifie le gradient des nœuds ainsi que leur voisinage, il faut donc mettre à jour le SCV en conséquent. Nous proposons donc un protocole de mise à jour qui fonctionne de manière localisé et qui effectue des mises à jour seulement dans les portions du réseaux touchés par les changements et seulement lorsque du trafic est acheminé. Nous montrons que cette solution offre un meilleur compromis taux de livraison/énergie consommée que les solutions de mise à jour périodique.

4.1 Système de coordonnées virtuelles

4.1.1 *Offset* : affiner la précision du gradient

Le SCV que nous proposons est composé d'une seule coordonnée qui représente la distance logique au puits. Cette coordonnée est un affinement du gradient : elle est calculée à partir du nombre de sauts pour atteindre le puits, mais elle positionne aussi le nœud dans l'anneau théorique formé par le gradient. Ce positionnement dans l'anneau de gradient est fonction de la répartition du voisinage du nœud dans les anneaux du gradient : les nœuds qui ont proportionnellement plus de voisins dans l'anneau plus proche du puits que le leur doivent être classés avant ceux qui en ont proportion-

nellement moins. Cette information peut être utilisée pour le routage comme dans [Romdhani, 2012] mais aussi pour discriminer les nœuds dans le 2-voisinage.

Comme illustré sur la Figure 4.2, nous nommons *offset* l'affinement du gradient (positionnement du nœud dans l'anneau de gradient). Notre démarche pour calculer cet *offset* est la suivante :

- dans un premier temps, à partir d'un modèle théorique (présenté dans la section suivante), nous exprimons la relation entre l'*offset* et les aires de recouvrement des anneaux de gradient et de la portée radio du nœud. Ces aires : A, B et C sont illustrées par la Figure 4.2 ;
- dans un second temps, nous établissons une correspondance entre ces aires et la répartition du voisinage d'un nœud dans les anneaux. De cette manière on peut calculer l'*offset* à partir de la répartition des voisins du nœud dans les différents anneaux du gradient (obtenue lors de l'initialisation du réseau détaillée dans la section 4.1.4).

Les sections suivantes détaillent ces deux étapes. La coordonnée est construite en utilisant le gradient (noté h) et l'*offset* comme illustré par la Figure 4.2, avec R la portée radio d'un nœud. La coordonnée est calculée de la manière suivante :

$$coord_p = (h-1) * R + offset \text{ avec } 0 \leq offset < R \tag{4.1}$$

4.1.2 Evaluation théorique de l'*offset*

Dans cette section, nous établissons la relation entre les aires A, B et C représentées sur la figure 4.2 et l'*offset*. Dans la section suivante, nous établissons la relation entre les aires et la répartition des voisins d'un nœud dans les anneaux de gradient.

Notre modèle théorique est basé sur le modèle *Unit disk Graph* (UDG) [Clark et al., 1990] ; cette hypothèse est relâchée lors de la mise en œuvre des coordonnées. En plus de l'UDG, nous supposons que les nœuds sont répartis de manière uniforme sur un plan, les nœuds possédant le même gradient forment alors des anneaux concentriques (centrés sur le puits) parfaits comme représentés sur la Figure 4.2. Dans ce cas, on observe que l'*offset* est directement lié aux valeurs des aires A, B et C. On peut donc trouver des fonctions du type $f(offset) = A$, $g(offset) = B$ et $h(offset) = C$. Ce lien est réalisé en calculant les aires en fonction de la position du nœud dans l'anneau h. L'aire A en fonction de l'*offset* est donnée par l'intégrale suivante :

$$A = \int_{\theta_1}^{\theta_2} \int_{r(\theta)}^{(h-1)R} r dr d\theta \tag{4.2}$$

avec

$$r(\theta) = [(h-1)R + offset]sin\theta - \sqrt{R^2 - [(h-1)R + offset]^2 cos\theta^2}$$

On pose

$$A_1 = \int_{r(\theta)}^{(h-1)R} r dr = \left[\frac{r^2}{2}\right]_{r(\theta)}^{(h-1)R}$$

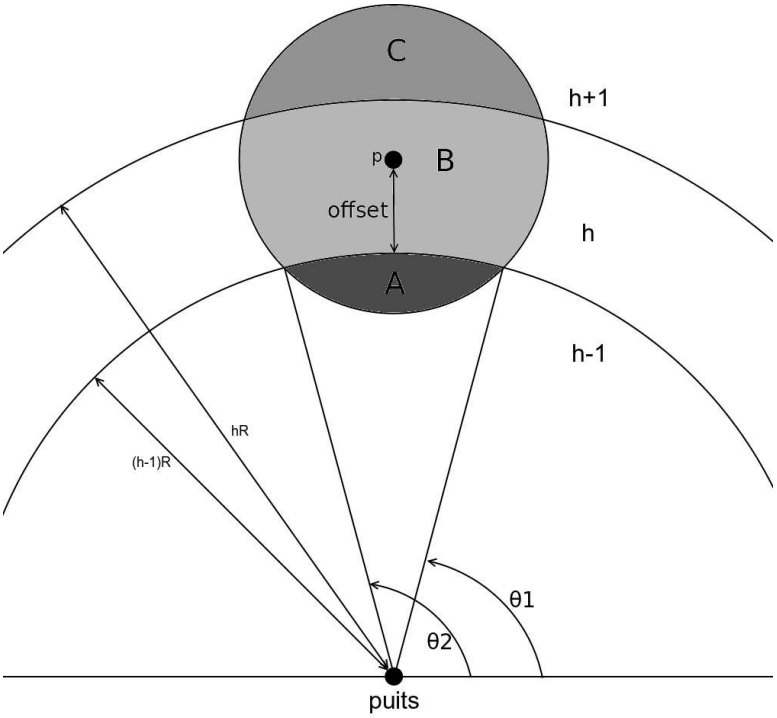


FIGURE 4.2 – Illustration du modèle théorique

donc

$$A = \int_{\theta_1}^{\theta_2} A_1 d\theta$$

$$\begin{aligned} A &= \frac{(h^2-2h)R^2}{2}\left[\theta\right]_{\theta_1}^{\theta_2} + \frac{\left[(h-1)R+offset\right]^2}{2}\left[\sin\theta\cos\theta\right]_{\theta_1}^{\theta_2} \\ &+ \frac{\left[(h-1)R+offset\right]}{2}\left[-\cos\theta\sqrt{R^2-\left[(h-1)R+offset\right]^2\cos^2\theta}\right. \\ &\left. - \frac{R^2\tan^{-1}\left(\frac{\left[(h-1)R+offset\right]\cos\theta}{\sqrt{R^2-\left[(h-1)R+offset\right]^2\cos^2\theta}}\right)}{(h-1)R+offset}\right]_{\theta_1}^{\theta_2} \end{aligned} \tag{4.3}$$

$$\text{avec} \begin{cases} \theta_1 = \arctan\left(\frac{(h-1)^2R^2-R^2+\left[((h-1)R+offset)\right]^2}{2\left[((h-1)R+offset)\right]}\right. \\ \left.\times\frac{+1}{\sqrt{(h-1)^2R^2-\left[\frac{(h-1)^2R^2-R^2+\left[((h-1)R+offset)\right]^2}{2\left[((h-1)R+offset)\right]}\right]^2}}\right) \\ \theta_2 = \arctan\left(\frac{(h-1)^2R^2-R^2+\left[((h-1)R+offset)\right]^2}{2\left[((h-1)R+offset)\right]}\right. \\ \left.\times\frac{-1}{\sqrt{(h-1)^2R^2-\left[\frac{(h-1)^2R^2-R^2+\left[((h-1)R+offset)\right]^2}{2\left[((h-1)R+offset)\right]}\right]^2}}\right)+\pi \end{cases}$$

De la même manière on peut calculer C :

$$C = \int_{\theta_1}^{\theta_2} \int_{hR}^{r'(\theta)} r dr d\theta \tag{4.4}$$

avec

$$r'(\theta) = [(h-1)R + offset]sin\theta + \sqrt{R^2 - [(h-1)R + offset]^2 cos\theta^2}$$

Le développement de l'expression 5.7 est similaire à celui de A (4.3) et n'est donc pas détaillé ici. B est donné par $\pi R^2 - A - C$.

Dans la section suivante, nous faisons correspondre des nombres de voisins et des aires. Pour que ces quantités soient comparables, nous utilisons des valeurs relatives : nous définissons donc les pourcentages des aires A, B et C par rapport à l'aire de la portée radio et les pourcentages de voisins dans les anneaux de gradient par rapport au nombre total de voisins. En divisant les expressions des aires A, B et C par $B = \pi R^2$ (l'aire de la portée radio) nous obtenons les expressions de l'*offset* en fonction des pourcentages d'aires. Le principe de notre proposition est ensuite de faire correspondre les pourcentages de voisins (répartition des voisins dans les anneaux de gradient) avec les pourcentages d'aires et donc d'être capable de fournir un *offset* à chaque nœud.

On peut noter que deux nœuds dans le même 2-voisinage ayant les mêmes pourcentages de nœuds dans chaque anneau obtiennent la même coordonnée. Cette situation est appelée collision. Dans la section 4.2 une analyse théorique et une évaluation par simulation montrent que, même si cette situation apparaît, elle reste relativement rare.

4.1.3 Application aux RCsF

D'une part, nous avons maintenant des fonctions qui relient les pourcentages des aires avec l'*offset*. D'autre part, nous supposons que chaque nœud connaît son gradient h et le pourcentage de ses voisins qui sont dans les anneaux de gradient $h-1$, h et $h+1$. Le protocole qui permet l'obtention de ces informations se base sur une découverte du voisinage et est détaillé dans la section 4.1.4. Pour obtenir une relation entre l'*offset* et les pourcentages de voisins on doit donc relier ces pourcentages avec ceux des aires. Ceci est réalisé grâce à une projection des pourcentages de voisins sur les pourcentages d'aires : un point dans l'espace des pourcentages de voisins est projeté sur le point qui lui est le plus proche dans les pourcentages d'aires. Dans la suite de ce chapitre, les pourcentages de voisins dans les anneaux de gradient $h-1$, h et $h+1$ sont notés $\%(h-1)$, $\%h$ et $\%(h+1)$; les pourcentages de surfaces sont notés $\%A$, $\%B$ et $\%C$.

On note d'abord, qu'en réalité, un nœud peut avoir des pourcentages de voisins qui ne correspondent pas à des valeurs possibles de pourcentages d'aires. C'est le cas, par exemple, si un nœud de gradient h n'a aucun voisin de gradient h, car on constate sur la Figure 4.2 que l'aire B n'est jamais nulle pour $0 \leq offset < R$. Cela implique que l'espace des valeurs des pourcentages de voisins est plus grand que celui des pourcentages d'aires. La Figure 4.3 représente le plan des pourcentages de voisins décrit par l'équation $\%(h-1) + \%h + \%(h+1) = 1$ et la courbe des pourcentages

Symbole	Signification
p_1 et p_2	Points dans l'espace des valeurs de pourcentages de voisins
$offset_1$ et $offset_2$	Respectivement l'*offset* de p_1 et p_2 après la projection mais avant l'addition de la distance de projection.
d_1 et d_2	Respectivement les distances de projection de p_1 et p_2
$\Delta offset$	Distance entre deux valeurs consécutives pour un espace d'*offset*s discret

TABLE 4.1 – Notations utilisées pour le Théorème 4.1.1

d'aires contenue dans ce plan. La projection des valeurs du plan sur la courbe fait que des nœuds qui ont des pourcentages de voisins différents se retrouvent avec les mêmes pourcentages d'aires et donc le même *offset* : cela signifie que la projection induit des collisions de coordonnées. Sur la Figure 4.3, les points p et q ont des positions différentes sur le plan (ils ont des répartitions des voisins dans les anneaux de gradient différentes) mais sont projetés sur le même point de la courbe et donc obtiennent le même *offset*. On limite le nombre de collisions dues à la projection en ajoutant la distance euclidienne de la projection (notée d) à l'*offset* :

$$offset' = offset + d \text{ et } coord = (h-1) * R + offset' \tag{4.5}$$

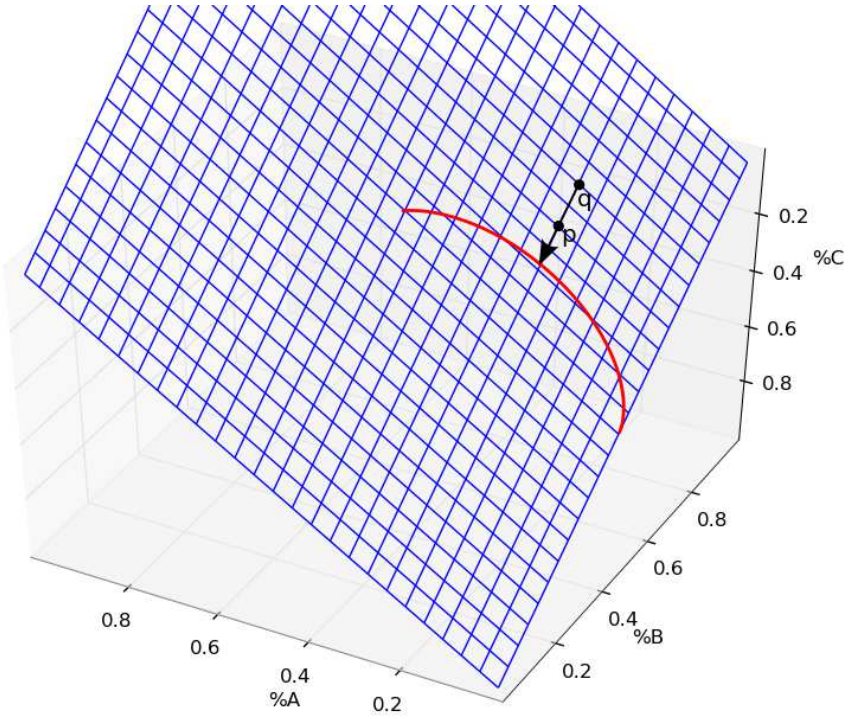


FIGURE 4.3 – Représentation du plan des valeurs possibles de répartition des voisins $(\%(h-1) + \%h + \%(h+1) = 1)$ et de la courbe reliant $\%A$, $\%B$ et $\%C$

Le théorème 4.1.1 établit que l'ajout de la distance à l'*offset* (cf. Equation 4.5) ne rajoute pas de collisions de coordonnées.

Théorème 4.1.1 *L'addition de la distance de projection à l'*offset *permet de réduire le nombre de collisions (sans en introduire de nouvelles) si l'espace des* offset *est discret et que deux valeurs d'*offsets *sont séparées au moins par la distance maximum de projection ($\Delta offset \geq d$).*

Preuve Nous raisonnons par l'absurde. Les notations sont détaillées dans le Tableau 4.1. Supposons que l'addition de la distance de projection crée une collision. Cela signifie que deux points (p_1 et p_2), qui n'ont pas obtenu le même *offset* (par exemple, $offset_1$ et $offset_2$ avec $offset_1 < offset_2$) obtiennent le même $offset'$ à cause de l'addition des distances de projection (d_1 et d_2) (cf. Equation 4.5). Cela est possible si $d_1 = offset_2 + d_2 - offset_1$. Or nous savons que $offset_2 - offset_1 \geq \Delta offset$, car $\Delta offset$ est la distance séparant deux valeurs consécutives d'*offset*, donc $offset_2 - offset_1 + d_2 > \Delta offset$ et nous concluons que $d_1 > \Delta offset$, ce qui est une contradiction. ∎

En pratique, un nœud peut embarquer une table qui contient les valeurs discrètes de la courbe $f(\%A, \%B, \%C) = offset$, obtenues dans la section 4.1.2. Ce nœud récupère les pourcentages de ces voisins dans les anneaux du gradient et projette ce point sur le point le plus proche dans la table. L'offset correspondant à ce point est donné au nœud, après y avoir ajouté la distance de projection. Cette technique permet d'avoir un jeu de données à faible granularité, car l'addition de la distance de projection évite les collisions de coordonnées. Cette propriété est intéressante dans le cas des RCsF car les nœuds ont généralement une mémoire de taille limitée.

4.1.4 Protocole de construction

Dans les sections précédentes, nous supposons que les nœuds connaissent leur gradient h ainsi que la répartition de leurs voisins dans $h-1$, h et $h+1$. Nous présentons dans cette section un protocole d'initialisation de la coordonnée qui permet à chaque nœud de récupérer ces informations.

Au démarrage du réseau, les nœuds ne sont pas synchronisés. Ils alternent les endormissements et les réveils, ce qui permet de réduire la consommation d'énergie [Polastre et al., 2004].

Le puits initie le protocole, il envoie le premier paquet d'initialisation qui contient le paramètre h (0 pour le puits). L'algorithme est décrit pour un nœud de gradient h. On note que les nœuds se synchronisent grâce à des préambules longs [Polastre et al., 2004] :

1. les nœuds de gradient $h-1$ émettent un préambule, cela permet de synchroniser les nœuds de gradient h ;
2. il y a ensuite une période de contention découpée en *slots* de temps. Chaque nœud de gradient $h-1$ choisit un *slot* de manière aléatoire uniforme et émet un paquet qui contient son numéro de gradient dans *slot* choisi ;

3. ces paquets permettent aux nœuds de gradient h de connaître le numéro de leur gradient (le numéro qu'ils reçoivent plus 1). En tirant parti du phénomène de sur-écoute (un seul paquet suffit pour établir le gradient mais un nœud en reçoit de nombreux), les nœuds peuvent compter le nombre de paquets reçus pendant la période de contention et ainsi connaître le nombre de voisins de gradient $h-1$ qu'ils possèdent ;
4. à la fin de la période de contention, ce processus est répété par les nœuds de gradient h ;
5. et par la suite il est répété par les nœuds de gradient $h+1$. Ainsi, après trois périodes de contention un nœud connaît son gradient et le nombre de voisins qu'il a à $h-1$, h et $h+1$.

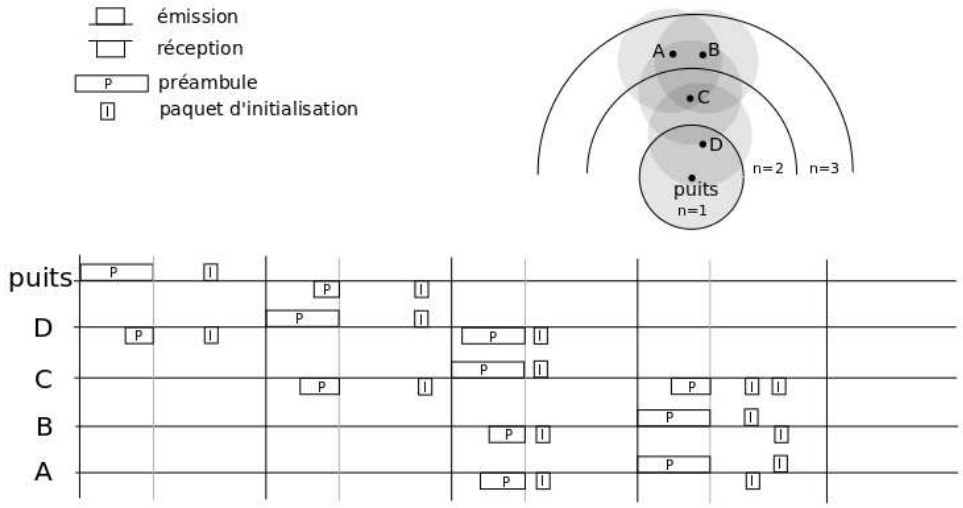


FIGURE 4.4 – Exemple d'initialisation des coordonnées.

La Figure 4.4 représente le protocole d'initialisation des coordonnées pour quatre nœuds et le puits. Le puits démarre le processus en envoyant un préambule. Le nœud D se réveille et, détectant la fin du préambule, il reste éveillé. Durant la période de contention, D reçoit un paquet du puits qui lui indique qu'il a un gradient de 1. A la fin de la première période de contention, D émet un préambule, cela permet de maintenir éveillé le nœud C pour la deuxième période de contention. D choisit un *slot* aléatoirement et envoie son paquet d'initialisation. De cette manière, C sait que son gradient est 2 et qu'il a un voisin de gradient 1, D sait qu'il n'a pas de voisin de gradient 1. C'est ensuite au tour de C d'émettre un préambule, puis un paquet d'initialisation. Après la troisième période de contention, les nœuds A et B connaissent le numéro de leur gradient, C sait qu'il n'a pas de voisin de gradient 2 et D qu'il a en a un seul. Enfin, A et B émettent un préambule simultanément et choisissent chacun un *slot* dans la dernière période de contention. Après cette dernière période, C sait qu'il a deux voisins de gradient 3 et A et B qu'ils en ont 1 chacun. On

remarque que les nœuds A et B, ayant la même répartition de voisins et étant dans le même 2-voisinage, sont en collision.

On peut noter qu'il est possible d'utiliser la période d'initialisation pour propager une valeur d'horloge dans le réseau et ainsi synchroniser les nœuds de proche en proche. Dans le cas où les nœuds sont déjà synchronisés avant l'exécution du protocole d'initialisation, les préambules sont inutiles. On peut aussi noter qu'avec ce protocole, les seules informations qu'un nœud doit stocker sont : son gradient h et le nombre de ses voisins à $h-1$, h et $h+1$. Cela donne une occupation mémoire constante quel que soit le degré du réseau, ce qui permet de passer à l'échelle.

Dans la section suivante, nous évaluons le nombre de collisions de coordonnées pour déterminer la capacité de notre proposition à discriminer les nœuds dans un 2-voisinage. En théorie, les collisions ont deux sources : soit les nœuds ont le même pourcentage de voisins dans chaque anneau, soit ils sont projetés sur le même point théorique avec la même distance. En pratique, les calculs des pourcentages et des distances de projection sont réalisés avec une précision finie, cela provoque de nouvelles collisions. Nous analysons ces cas dans les sections suivantes.

4.2 Évaluation du SCV

4.2.1 Evaluation théorique

Dans cette section, nous analysons la probabilité d'observer des collisions de coordonnées. Pour cela, nous caractérisons l'espace des coordonnées possibles dans le voisinage d'un nœud et nous calculons l'espérance du nombre de paires de nœuds qui obtiennent la même coordonnée et qui sont dans le voisinage d'un même nœud. Nous supposons que les nœuds sont distribués de manière aléatoire et uniforme sur un plan et qu'un nœud de gradient h a toujours au moins un voisin à $h-1$ ($\%(h-1) > 0$) (c'est-à-dire que le gradient est bien construit : si on a pas de voisin à $h-1$ c'est qu'on se situe au moins à $h+1$ et non à h). Comme on ne fait pas d'hypothèse sur la répartition du voisinage d'un nœud, on suppose que la coordonnée est choisie aléatoirement de manière uniforme dans l'espace des coordonnées possibles. Cette dernière hypothèse implique que l'on suppose que les positions des voisins des nœuds d'un même voisinage sont indépendantes. Intuitivement, ce n'est pas le cas car deux nœuds très proches géographiquement vont avoir tendance à avoir des voisinages assez similaires. Cette supposition va donc permettre à plus de nœuds d'obtenir des coordonnées différentes. De plus, nous ne prenons pas en compte les collisions dues aux projections dans cette partie, cela signifie que seuls les nœuds qui ont exactement les mêmes proportions de voisins sont en situation de collision. Ces hypothèses impliquent que nous prenons en compte moins de collisions dans l'analyse théorique que celles qui apparaissent en réalité. Cependant, cette analyse nous permet de comprendre qualitativement comment évolue le nombre de collisions quand le degré du réseau varie. Pour obtenir une estimation quantitative plus réaliste du nombre de collisions, nous évaluons ce paramètre par simulation dans la section 4.2.2.

Caractérisation de l'espace des coordonnées

Tout d'abord, nous caractérisons l'espace des coordonnées. Les nœuds peuvent se différentier grâce à leurs proportions voisins de gradients $h-1$, h et $h+1$. Nous considérons un nœud avec k voisins qui ont aussi tous k voisins. Les combinaisons possibles de proportions de voisins définissent l'espace des coordonnées. Le nombre de proportions accessibles dépend du nombre de voisins. Si un nœud a 2 voisins, il peut en avoir 0% ou 50% dans l'anneau h (100% n'est pas possible car il y à au moins un voisin dans l'anneau $h-1$), s'il a 3 voisins, il peut y avoir 0%, 33% ou 66% dans l'anneau h. Le cardinal de l'espace des coordonnées (noté C) augmente avec le nombre de voisins. La possibilité d'avoir une proportion de voisins avec un gradient donné dépend des proportions dans les autres gradients : par exemple, si un nœud a 3 voisins et qu'il en a 33% dans l'anneau $h-1$, il peut en avoir 33% dans l'anneau h et 33% dans l'anneau $h+1$, ou 66% dans h et 0% dans $h+1$, ou encore 0% dans h et 66% dans $h+1$, il n'y pas d'autres possibilités.

Nous avons donc $\%(h-1)+\%h+\%(h+1)=1$ qui peut être écrit :

$$m\frac{1}{k}+o\frac{1}{k}+p\frac{1}{k}=1 \tag{4.6}$$

avec $m+o+p=k$, et k le nombre de nœuds dans un voisinage, $m\in[1,k]$ et o, $p\in[0,k]$ respectivement le nombre de voisins dans $h-1$, h et $h+1$. Si m est fixé nous avons $k-m=o+p$ donc

$$\left.\begin{array}{ll} & o=(k-m) \text{ et } p=0 \\ \text{ou bien} & o=(k-m-1) \text{ et } p=1 \\ \ldots & \\ \text{ou bien} & o=0 \text{ et } p=(k-m) \end{array}\right\} k-m+1 \text{ possibilités}$$

Nous sommons le nombre de possibilités pour chaque valeur de m :

$$\sum_{m=1}^{k} k-m+1 = k+(k-1)+\ldots+1=\frac{k(k+1)}{2}$$

$$C=\frac{k(k+1)}{2} \tag{4.7}$$

Espérance du nombre de collisions

Nous voulons obtenir l'espérance du nombre de collisions en fonction du nombre de voisins des nœuds du réseau (noté k). Si nous supposons que les coordonnées sont tirées de manière uniforme dans l'espace de coordonnées, dans un voisinage de k nœuds, la probabilité qu'un nœud donné i ait la même coordonnée qu'un nœud j est donnée par $\frac{1}{C}$. Posons X_{ij}, une variable aléatoire tel que $X_{ij}=1$ s'il y a une collision entre i et j et $X_{ij}=0$ sinon, la variable aléatoire $X=\sum_{i\neq j} X_{ij}$ représente donc le nombre de paires de nœuds en collision pour un réseau de degré k. On note ici qu'en comptant le nombre de paires de nœuds en collisions on surestime le nombre

de collisions : prenons trois nœuds a, b et c qui ont la même coordonnée (ils sont en collision), en comptant le nombre de paires de nœuds en collision on compte la collision entre a et b, a et c et b et c donc trois collisions au lieu d'une. On a dans ce cas l'espérance du nombre de collisions $E[X] = \sum_{i \neq j} E[X_{ij}]$:

$$E[X] = \begin{pmatrix} k \\ 2 \end{pmatrix} \frac{1}{C} = \frac{k!}{2!(k-2)!} \frac{2}{k(k+1)} = \frac{k-1}{k+1} \tag{4.8}$$

car $C = k(k+1)/2$. On note que

$$\lim_{k \to +\infty} \frac{k-1}{k+1} = 1 \tag{4.9}$$

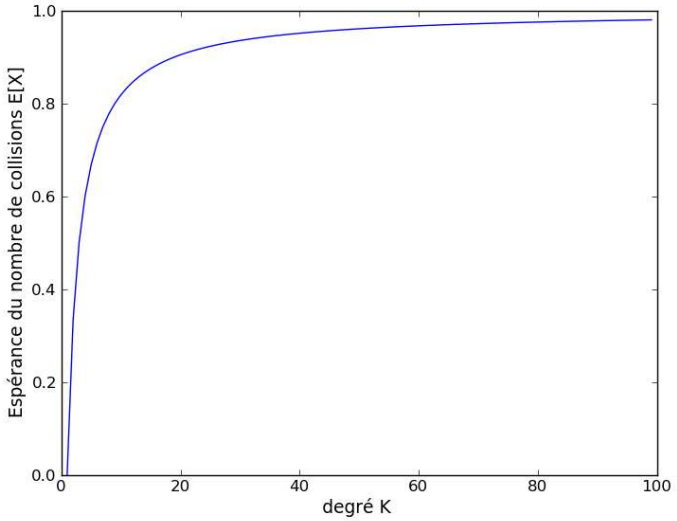


FIGURE 4.5 – Espérance du nombre de collisions en fonction de la taille d'un voisinage

La Figure 4.5 représente un tracé de l'équation 4.8 en fonction du nombre de voisins k. La courbe reste toujours en-dessous de la valeur 1, $E[X]$ est bornée par la valeur 1. L'espérance du nombre de collisions dans un 2-voisinage ne dépend donc pas du degré moyen du réseau. Cependant, en réalité il y a plus de collisions en moyenne comme nous le montrons dans la section 4.2.2. C'est dû aux collisions induites par la projection (cf. section 4.1.3), au fait que les répartitions des voisins des nœuds qui sont voisins ne sont pas indépendantes et aussi au fait que l'implémentation utilise des coordonnées à précision finie. Ce résultat reste tout de même intéressant, car il montre que le nombre de collisions devrait rester stable quand le degré du réseau augmente.

4.2.2 Evaluation par simulations

Dans cette section, nous évaluons par simulation le nombre de collisions obtenues après une initialisation du réseau. De cette manière, nous confirmons le résultat de la section théorique (le nombre de collisions reste stable quand le degré du réseau augmente), tout en obtenant des valeurs plus réalistes. Pour réaliser les simulations, nous utilisons le simulateur à événements discret WSNet [WSNet, 2009]. Ce simulateur permet d'évaluer les réseaux sans fil large échelle, il implémente notamment de nombreux modèles de propagation (*two-ray ground*, *nakagami*, *log-normal shadowing*, etc). Nous simulons un réseau dans une aire de 50×50 unités de surface avec le puits à $(25, 25)$, le rayon de communication est de 10 unités (on choisit une aire relativement petite car cela permet de limiter les durées de simulation : on peut avoir une forte augmentation du degré, en ayant une augmentation relativement faible du nombre de nœuds). Les simulations sont effectuées avec deux modèles de propagation différents. Le modèle de pertes en espace libre (*free space* [Goldsmith, 2005]) qui ne prend pas en compte le phénomène de *fading*, et le modèle *log-normal shadowing* [Zuniga and Krishnamachari, 2004] présenté dans le Chapitre 3. Dans les deux cas, les interférences des autres nœuds sont prises en compte. Nous simulons le protocole d'initialisation précédemment décrit avec un nombre de nœuds variant de 50 à 750 placés aléatoirement sur une aire de 50x50 unités de longueur (pour chaque nombre de nœuds on effectue 20 simulations), cela permet de faire varier la densité du réseau et donc le degré car la puissance de transmission reste constante. Cela produit des réseaux avec entre 3 et 5 sauts maximum pour atteindre le puits en fonction des topologies.

Le but de cette évaluation de performances est d'évaluer le nombre de collisions de coordonnées résultant de la méthode de construction que nous proposons. Dans le simulateur, les coordonnées sont stockées sous forme de nombres à virgule flottante avec une précision finie, cela peut provoquer des collisions. Ces collisions ne proviennent pas directement de notre méthode, mais elles doivent être prises en compte car elles apparaissent nécessairement lors de l'implémentation.

Modèle de pertes en espace libre

La figure 4.6 représente le nombre moyen de collisions dans le voisinage d'un nœud en fonction de son degré. Pour un nœud donné nous comptons le nombre de fois où plusieurs de ses voisins ont la même coordonnée. Ce nombre reste stable quand le degré du réseau augmente, cela confirme le résultat théorique de la section 4.2.1. Cependant, le nombre moyen de collisions est plus important que celui prédit dans l'analyse théorique. Cela est dû aux hypothèses décrites dans la section 4.2.1 ainsi qu'aux sources de collisions que nous ne prenons pas en compte dans le modèle théorique (précision finie, projection, etc). À partir de ces résultats, on peut déduire que notre solution est plus performante dans les réseaux de fort degré car un nœud a moins de voisins en collision en proportion : le nombre moyen de collisions dans le voisinage d'un nœud est environ 2, si l'on fait l'hypothèse que les collisions impliquent seulement des paires de nœuds. Cela signifie qu'en moyenne un nœud à 4 voisins en

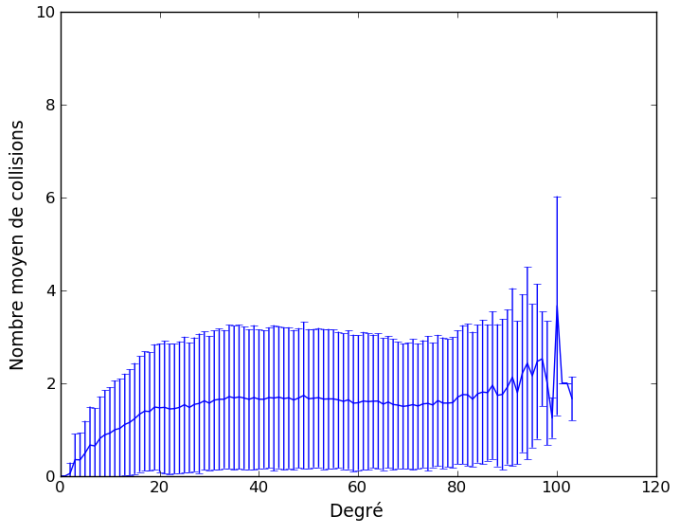


FIGURE 4.6 – Nombre moyen de collisions (les barres d'erreurs correspondent à deux fois l'écart type) en fonction de la densité du réseau pour le modèle pertes en espace libre.

situation de collision. Donc pour un voisinage de 10 nœuds cela fait 40% des voisins en collision et pour un voisinage de 100 nœuds, cela fait 4% des voisins.

Modèle *log-normal shadowing*

La figure 4.7 représente le nombre moyen de collisions dans le voisinage d'un nœud en fonction de son degré avec le modèle de propagation *log-normal shadowing*. Dans ce cas, le nombre de collisions est légèrement moins important que dans le cas du modèle de pertes en espace libre. C'est dû au fait que le modèle de propagation ajoute de l'aléa dans la composition des voisinages. De cette façon, des nœuds voisins ont moins de chances d'avoir la même répartition de voisins. On constate que le nombre de collisions, même s'il augmente légèrement, reste assez stable quand le degré du réseau augmente.

Comme évoqué précédemment, ces collisions sont un problème car on veut pouvoir discriminer les nœuds dans un 2-voisinage. D'un autre côté, il y a peu de collisions, on constate qu'au moins 95% des valeurs entre 20 et 90 voisins sont en-dessous de 3 avec les deux modèles de propagation.

4.3 Maintenance des coordonnées

Au cours de la vie du réseau, la topologie peut évoluer pour plusieurs raisons :

- des nœuds qui n'ont plus de batterie ;
- des nœuds détruits par un phénomène naturel (feux par exemple) ;
- des nœuds sont ajoutés au réseau ;

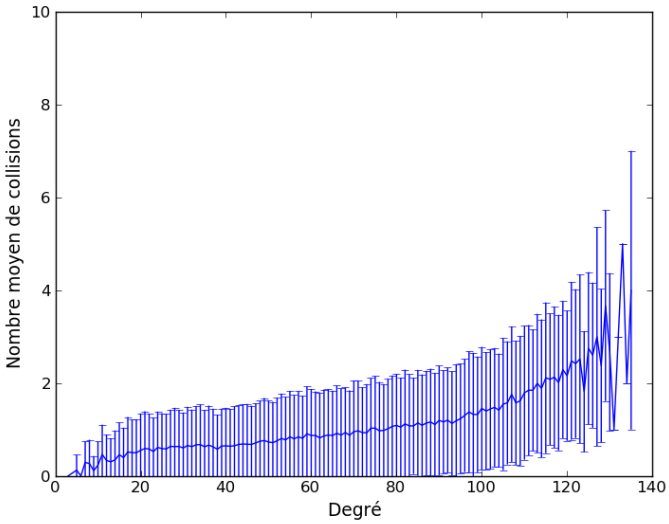


FIGURE 4.7 – Nombre moyen de collisions (les barres d'erreurs correspondent à deux fois l'écart type) en fonction de la densité du réseau pour le modèle *log-normal shadowing*.

- des liens qui disparaissent de façon permanente (objets non mouvants ajoutés à l'environnement).

Ces événements peuvent modifier le nombre de sauts qui séparent un nœud du puits ainsi que la répartition des voisins et ainsi modifier complètement la coordonnée. Si le SCV n'est pas remis à jour lors de changements dans le réseau, il ne fournit plus une vision cohérente du réseau, cela dégrade les performances d'accès au médium (délais, collisions) et du routage (longueur de routes, taux de livraison).

Pour mettre à jour le SCV, plusieurs alternatives sont possibles :

- Réinitialisation périodique des coordonnées [Watteyne et al., 2009b] suivant le protocole présenté dans la section 4.1.4. En fonction de la dynamique de changement dans la topologie, une période est déterminée ; le puits relance donc régulièrement l'initialisation du réseau. Les principaux inconvénients de cette méthode sont que le réseau dans son ensemble est réinitialisé même si certaines parties ne sont pas touchées par le changement, cela induit une surconsommation énergétique, et que la dynamique de changement de topologie doit être connue à l'avance pour déterminer la fréquence de mise à jour. De plus, durant la réinitialisation du réseau, les paquets de données ne peuvent pas être acheminés, cela augmente les délais de bout en bout.
- Réinitialisation déclenchée par un événement, c'est la méthode utilisée dans [Ye et al., 2005a] : au dessous d'un seuil de taux de livraison de paquets, une nouvelle initialisation des coordonnées est déclenchée. Dans ce cas, la dynamique de changement dans le réseau n'a pas besoin d'être connue à l'avance mais le réseau est réinitialisé même si seulement une partie est touchée par les changements de topologie. Dans ce cas aussi on retrouve les problème de surcon-

sommation d'énergie et de délai. On remarque qu'en pratique, cette méthode est difficile à mettre en œuvre : pour pouvoir déclencher la mise à jour, il faut connaître le taux de livraison, il faut donc que le puits connaisse le nombre de paquets émis. Pour une application de détection d'événement, on ne sait pas à l'avance quels nœuds vont émettre, le puits ne peut donc pas calculer le taux de livraison.

- Réinitialisation déclenchée par une utilisation des coordonnées : les coordonnées sont utilisées pour router des informations. Lorsqu'un nœud de gradient h veut router un paquet en direction du puits, il doit avoir au moins un voisin plus proche du puits que lui en terme de nombre de sauts, sinon cela veut dire que son gradient est h et non $h+1$ (l'absence de voisin plus proche du puits signifie qu'il y a un trou dans le réseau). La coordonnée peut donc être mise à jour au moment du routage seulement si aucun nœud plus proche de l'émetteur du paquet ne répond. C'est l'approche adoptée dans SGF [Huang et al., 2009]. Dans la section suivante, nous détaillons cette approche, montrons ses limitations et proposons un protocole qui permet de les lever. Ce protocole est un protocole de mise à jour de gradient. Pour mettre à jour la coordonnée virtuelle présentée dans ce chapitre, il faut aussi mettre à jour la répartition des voisins. Nous détaillons comment réaliser cette mise à jour en tirant parti de la sur-écoute durant la mise à jour du gradient.

Mise à jour du gradient

Dans [Watteyne et al., 2009b], les auteurs affirment que la plupart des solutions de mise à jour de gradient de la littérature sont basées sur une phase de d'initialisation exécutée périodiquement. Il n'est cependant pas indiqué dans la littérature, quelle doit être la fréquence de mise à jour ni comment faire cohabiter le trafic de paquets de donnée et le trafic issu des mises à jour [Watteyne et al., 2009b]. On peut noter que les auteurs de [Zhao et al., 2007] [Gnawali et al., 2008] et [Watteyne et al., 2009b] proposent un mécanisme de mise à jour du gradient basé sur des échanges de messages *hello* qui permettent aussi de mettre à jour les voisinages des nœuds. Cependant, comme évoqué dans la section 2.1.2 du Chapitre 2, ce mécanisme de maintient de structure de routage induit une forte consommation d'énergie [Amadou, 2012] dans les réseaux de degré élevé.

SGF [Huang et al., 2009] est un protocole de routage opportuniste par gradient, les auteurs proposent de maintenir le gradient seulement quand des paquets de données circulent et qu'ils ne peuvent plus être routés : aucun voisin plus près du puits ne répond. Le routage dans SGF fonctionne de la même manière que dans GRAB [Ye et al., 2005a] (décrit en détails dans la section 2.1.2 du Chapitre 2). Il intègre en plus la prise en compte de l'énergie dans le coût pour atteindre le puits et la mise à jour des coûts lors du routage. Le principe de la mise à jour du gradient est le suivant : si un nœud n'a pas de voisin plus proche du puits que lui, il renvoie le paquet au nœud précédent et se retire du processus de routage, et ainsi de suite jusqu'à ce qu'un nœud ait un voisin plus proche du puits que lui. Ce mécanisme provoque des retours en arrière du paquet non nécessaires et dans certains cas, mène à une perte

de paquet alors qu'il existe un chemin vers le puits : c'est le cas, par exemple, avec le réseau illustré par la Figure 4.8. Si un paquet provenant de A atteint le nœud C après la mort du nœud D, le paquet va revenir au nœud B puis A et donc arriver dans une impasse alors qu'il existe un chemin pour atteindre le puits par H.

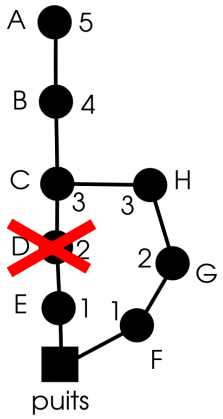


FIGURE 4.8 – Exemple d'échec de maintenance de gradient de SGF (les chiffres correspondent aux gradients des nœuds).

Pour palier à cette limitation de la maintenance de SGF, nous proposons un mécanisme de mise à jour de gradient localisé, déclenché par le trafic qui effectue la mise à jour seulement au moment, et dans la zone, où elle est nécessaire. Ce protocole est basé sur un mécanisme de routage opportuniste qui permet de router sans connaissance du voisinage et offre une meilleure fiabilité que le routage classique (cf. Chapitre 3). Le fonctionnement général du protocole que nous proposons est similaire à celui de GRAB sauf qu'on laisse la possibilité aux nœuds à égale distance ou plus éloignés du puits que l'émetteur d'acquitter et donc relayer les paquets, si aucun nœud plus proche ne répond. Comme illustré par la Figure 4.9, le paquet de données contient le gradient h de l'émetteur. La période d'acquittement est divisée en trois parties : la première pour les nœuds plus proches du puits que l'émetteur (qui sont donc les plus prioritaires), la deuxième pour les nœuds à égale distance du puits que l'émetteur et une pour les nœuds plus éloignés. Chaque partie correspond à une période de contention : les nœuds participants déclenchent une temporisation dont la durée dépend d'une métrique (énergie restante, qualité du lien, etc). Dans cette section, nous considérons qu'un nœud choisit sa temporisation de manière aléatoire et uniforme (un nœud à égale distance du puits que l'émetteur choisit une durée entre t_2 et t_3). Un nœud dont la temporisation expire envoie un paquet d'acquittement et devient le relayeur du paquet. Si un nœud reçoit un paquet d'acquittement avant la fin de sa temporisation, il perd la contention. Pour éviter le problème de duplication des paquets, nous utilisons deux types de paquets d'acquittement : l'acquittement (noté $ackN$) du relayeur est répété par l'émetteur (noté $ackS$). De cette manière, les nœuds voisins de l'émetteur, mais pas du relayeur, savent qu'ils sont perdants. Nous nommons notre protocole GRABUP (pour *GRAB UPdate*).

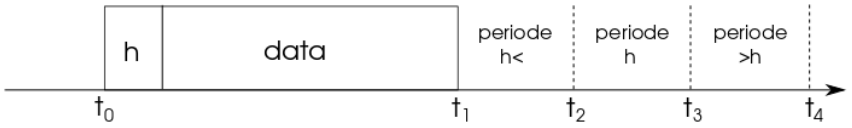


FIGURE 4.9 – Organisation du protocole de mise à jour : la période d'acquittement est divisée en trois parties

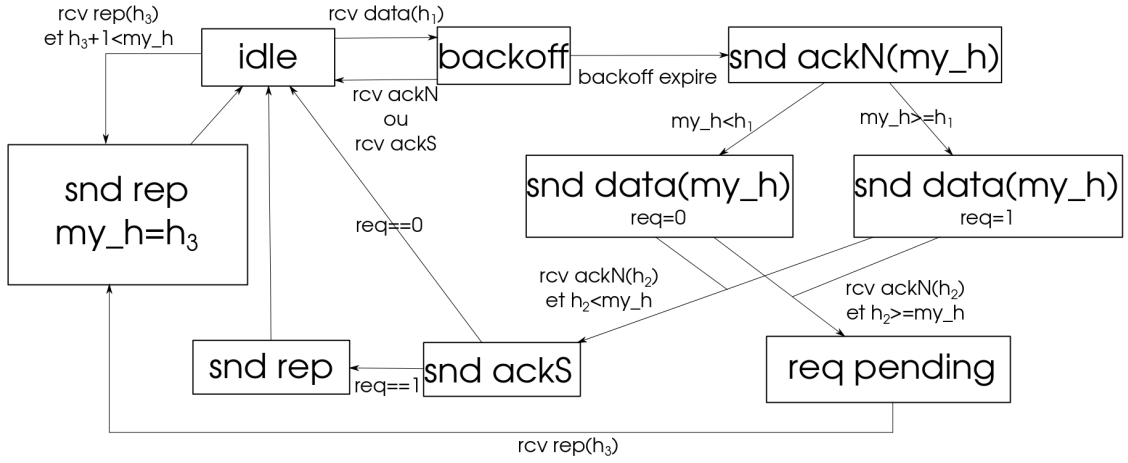


FIGURE 4.10 – Modèle à états représentant le fonctionnement du protocole de mise à jour de gradient.

La Figure 4.10 est un modèle à états qui représente le fonctionnement détaillé de notre proposition. Après l'initialisation du gradient (protocole de la section 4.1.4), les nœuds sont dans l'état *idle*.

- Quand un nœud reçoit un paquet de données avec h_1, il passe dans l'état *backoff* et choisit une durée de temporisation qui dépend de son gradient comme décrit précédemment (cf. Figure 4.9).
- Sur réception d'un *ackN* (acquittement d'un nœud relayeur) ou d'un *ackS* (acquittement du nœud émetteur), le nœud retourne dans l'état *idle* (il n'est pas relayeur et donc supprime le paquet).
- Si sa temporisation expire, il envoie un *ackN* qui signifie qu'il est élu relayeur du paquet.
- Ensuite, si $h_1 > my\_h$ (my_h étant le gradient du nœud), le nœud relaye le paquet avec son gradient my_h dans l'en-tête et sa variable *req* garde la valeur 0. La variable *req* prend la valeur 1 si $h_1 \leq my\_h$, cela signifie que le gradient h_1 de l'émetteur n'est pas à jour, car aucun nœud avec un gradient inférieur à l'émetteur n'a répondu : une mise à jour du gradient doit alors être déclenchée.
- Le nœud envoie ensuite le paquet de données et devient donc émetteur, il attend alors un *ackN*. En fonction du gradient h_2 du nœud qui envoie le *ackN*, le nœud envoie un *ackS* (si $h_2 < my\_h$) ou bien passe dans l'état *request pending* (si $h_2 \geq my\_h$).

Paramètre	Valeur
Débit des nœuds	500kbps
Taille des paquets de données	100 octets
Taille des paquets d'acquittement	10 octets
Nombre de nœuds	200 à 400
Aire de simulation	50×50 unités
Portée radio	10 unités
Puissance consommée en TX	65.7mW
Puissance consommée RX	53.7mW
Trafic de paquets de données (pour l'ensemble du réseau)	1 paquet/5s
Taux de mortalité des nœuds	1 nœud/5s
Durée des simulations	1000s

TABLE 4.2 – Paramètres de simulation

  - Dans le premier cas, le nœud envoie ensuite (après le *ackS*) un paquet *rep* si *req* est égal à 1 et retourne dans l'état *idle*, cela signifie qu'il a un gradient à jour et qu'il en informe des voisins non à jour.
  - Dans le second cas ($h_2 < my\_h$), le nœud attend un paquet *rep* pour remettre à jour son gradient (il ne prend alors plus part au processus de routage jusqu'à ce que son gradient soit à jour), quand il reçoit un paquet *rep* (qui contient l'ancien et le nouveau gradient du nœud qui l'émet), il met à jour son gradient et envoie à son tour un message *rep*. On note que dans ce dernier cas, le nœud n'envoie pas de *ackS* car cela permet de dupliquer le paquet pour explorer différente routes pour atteindre un nœud à jour et ainsi garder la route plus courte.
- Un nœud dans l'état *idle* qui reçoit un paquet *rep* met à jour son gradient et envoie un nouveau *rep* seulement si le nouveau gradient est plus petit que l'ancien.

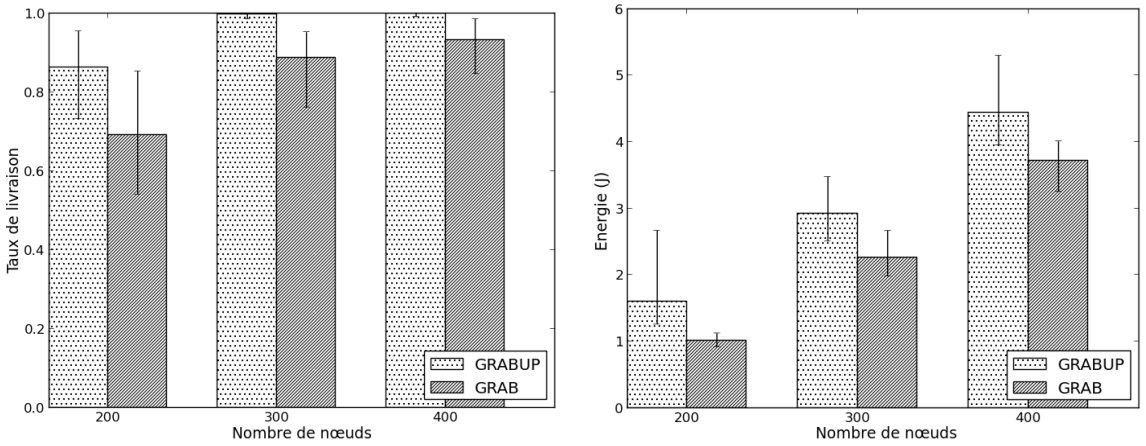
Dans la section suivante, nous évaluons par simulation le protocole de mise à jour de gradient proposé.

Performances de la mise à jour du gradient

Le but de cette section est de comparer les performances du routage opportuniste par gradient avec une mise à jour périodique, avec notre protocole de mise à jour GRABUP. Nous évaluons le taux de livraison, la consommation d'énergie et le délai de bout en bout de notre proposition et comparons ces paramètres avec ceux de GRAB [Ye et al., 2005a] pour montrer l'intérêt de notre méthode de mise à jour. Comme dans la section 4.2.2, les simulations sont réalisées à l'aide du simulateur à événements discrets WSNet [WSNet, 2009]. Nous faisons ici le choix de ne pas simuler la couche physique et la couche MAC, car ces couches affectent de la même manière GRAB et GRABUP qui utilisent les mêmes mécanismes (routage opportuniste, acquittements, etc).

Le Tableau 4.3 résume les paramètres de simulation. Les paramètres de consommation d'énergie sont tirés de [Lampin et al., 2012], comme nous ne simulons pas la couche MAC, l'énergie consommée provient seulement des échanges de paquets. L'énergie consommée pour envoyer ou recevoir un paquet est donnée par $E = P \times D$ avec P égale respectivement à la puissance consommée en TX ou RX et D la durée d'un paquet (l'énergie est donnée en Joules sur les figures). Nous avons généré des topologies aléatoires de 200, 300 et 400 nœuds, pour chaque nombre de nœuds, 20 topologies ont été produites (on effectue donc 20 simulations pour chaque nombre de nœuds). Cela nous permet de faire varier la densité et donc le degré du réseau (car la portée radio ne varie pas). La topologie évolue à cause de morts de nœuds : toutes les 5 secondes un nœud est choisi aléatoirement de manière uniforme parmi les nœuds vivants et est désactivé.

Dans un premier temps, nous comparons les performances (taux de livraison, énergie, délai de bout en bout) de GRABUP et de GRAB sans mise à jour périodique. Puis, dans un second temps, nous comparons les performances de ces deux solutions lorsque le gradient de GRAB est mis à jour périodiquement.

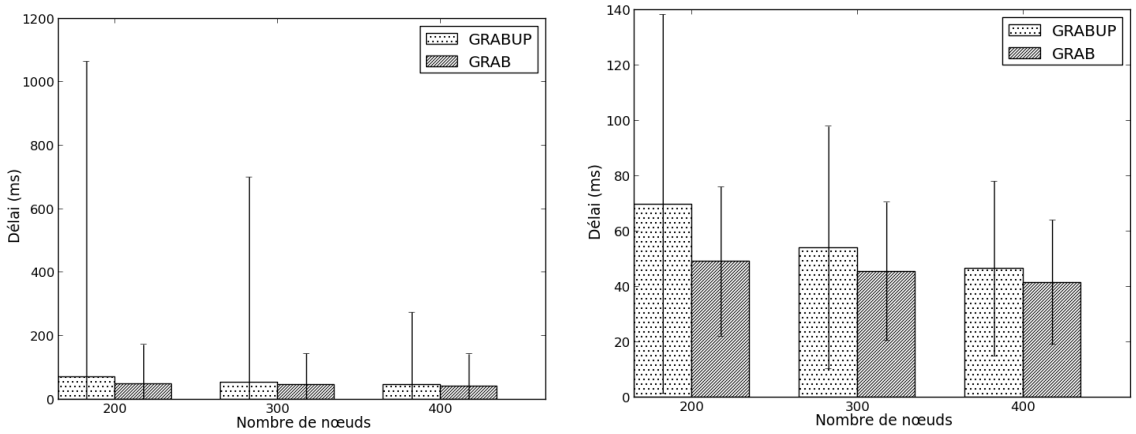


(a) taux de livraison (b) énergie

FIGURE 4.11 – Taux de livraison et consommation d'énergie pour des topologies de 200, 300 et 400 nœuds : valeurs moyennes (les barres d'erreur représentent les valeurs minimales et maximales).

La Figure 4.11(a) représente les résultats de comparaison des taux de livraison moyens de GRAB et de GRABUP pour des topologies de 200, 300 et 400 nœuds. Les barres d'erreur indiquent les valeurs minimales et maximales pour 20 topologies. On observe que notre solution a un taux de livraison plus élevé dans tous les cas évalués. Cependant, la différence de valeurs décroît quand le nombre de nœuds croît. C'est dû au fait que l'augmentation du nombre de nœuds induit une augmentation du degré, les morts de nœuds ont donc moins de chance de créer des trous dans le réseau et donc de modifier le gradient. L'observation des traces de simulation indique que les pertes de paquets de notre solution de mise à jour sont seulement dues à des topologies déconnectées.

La Figure 4.11(b) représente les résultats de comparaison des consommations d'énergie moyennes de GRAB et de GRABUP pour des topologies de 200, 300 et 400 nœuds (les barres d'erreur indiquent les valeurs minimales et maximales pour 20 topologies). Nous observons que la consommation d'énergie de notre proposition est plus importante que celle de GRAB, c'est dû à la partie de maintenance de notre algorithme et au fait que plus de paquets sont livrés au puits. La différence de consommation d'énergie décroît quand le nombre de nœuds croît, car moins de mises à jour de gradient sont nécessaires.



(a) barres d'erreur représentant les valeurs min-max

(b) barres d'erreur représentant l'écart type

FIGURE 4.12 – Délais de bout en bout pour des topologies de 200, 300 et 400 nœuds

La Figure 4.12(a) représente les résultats de comparaison des délais de bout en bout moyens de GRAB et de GRABUP pour des topologies de 200, 300 et 400 nœuds (les barres d'erreur indiquent les valeurs minimales et maximales pour 20 topologies). On remarque que les valeurs maximums de GRABUP sont très hautes. En se penchant sur les traces de simulation, on observe que ces délais sont en fait très rares et sont dus à des situations comme représentées sur la Figure 4.13. En effet, dans cette situation, si le nœud D envoie un paquet après la mort de A, le paquet est envoyé à C puis à B et revient ensuite en arrière pour atteindre le puits par l'autre côté. Dans ce cas le paquet fait 9 sauts au lieu de 5 (cas où le gradient est à jour). Cela permet cependant de livrer les paquets, ce qui n'est pas le cas avec GRAB.

La Figure 4.12(b) représente les mêmes résultats que 4.12(a) mais avec les barres d'erreur illustrant l'écart type. Cela permet une meilleure visibilité des différences de moyennes entre les deux solutions. On observe que la différence de délais moyens décroît quand le nombre de nœuds augmente car on rencontre moins de cas du type de celui représenté par la Figure 4.13.

Comme évoqué précédemment, la solution de mise à jour de gradient la plus utilisée dans la littérature est la mise à jour périodique par inondation [Watteyne et al., 2009b]. Nous appliquons cette solution à GRAB et nous nous intéressons à la période de rafraîchissement du gradient nécessaire pour avoir le même taux de livraison qu'avec GRABUP. Les simulations sont réalisées pour des topologies

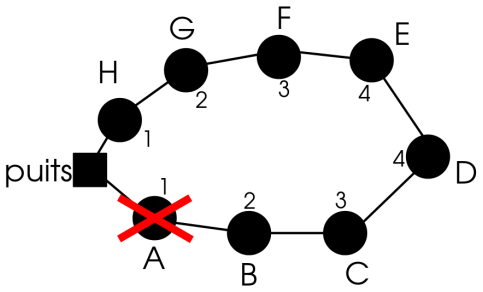
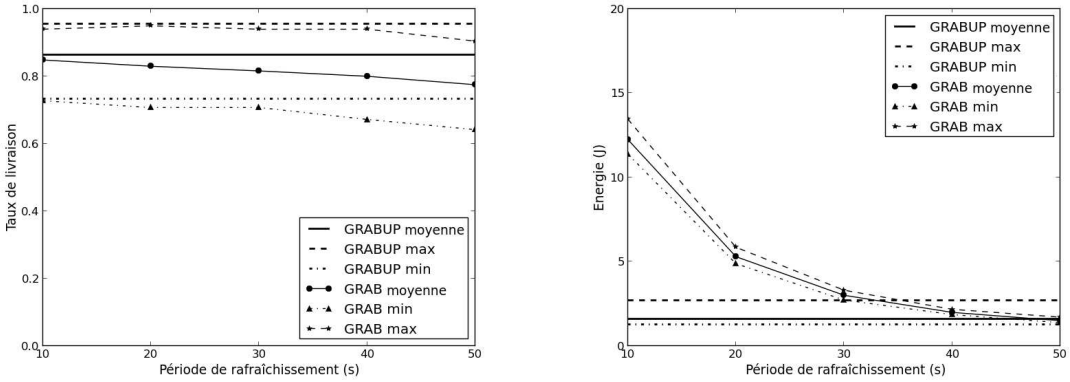


FIGURE 4.13 – Exemple de topologie qui induit de longs delais de bout en bout.

de 200 nœuds avec la période de rafraîchissement variant de 10 (correspondant à la plus grande fréquence) à 50 secondes.



(a) Taux de livraison

(b) Consommation d'énergie

FIGURE 4.14 – Simulations avec rafraîchissement du gradient périodique pour GRAB pour des topologies de 200 nœuds.

La Figure 4.14(a) représente les valeurs moyennes maximums et minimums du taux de livraison de GRAB en fonction de la période de rafraîchissement. Les résultats de GRABUP sont indiqués pour comparaison. Nous observons qu'il faut une période de rafraîchissement de 10 secondes pour que GRAB atteigne un taux de livraison proche de GRABUP. Pour des périodes plus importantes, les performances de GRAB se détériorent et sont inférieures à celle de GRABUP. Sur la Figure 4.14(b) on observe la consommation d'énergie en fonction de la période de rafraîchissement du gradient. On remarque d'abord que l'énergie consommée par GRAB décroît rapidement avec l'augmentation de la période de rafraîchissement. Pour des périodes inférieures à 40 secondes l'énergie consommée par GRAB est supérieure à celle consommée par GRABUP, pour des périodes supérieures à 40 secondes, elle est égale ou inférieure.

Sous les conditions de trafic et de mortalité de nœuds simulés, GRABUP permet un meilleur compromis entre énergie consommée et taux de livraison que GRAB avec un rafraîchissement périodique. C'est dû au fait que GRABUP met à jour le gradient seulement quand c'est nécessaire (quand un paquet est acheminé vers le puits) et où

c'est nécessaire (sur le chemin emprunté par le paquet), cela permet de concentrer les dépenses d'énergie où c'est nécessaire, tout en maintenant un taux de livraison élevé (les pertes apparaissent seulement dans les topologies déconnectées).

Mise à jour de l'*offset*

GRABUP permet de mettre à jour le gradient. Or la coordonnée virtuelle proposée dans ce chapitre, en plus de donner le gradient, affine cette information grâce à l'*offset*. Ce paramètre est calculé grâce à la répartition des voisins du nœud, l'apparition ou la disparition de liens ou de nœuds modifie la répartition des voisins et donc l'offset. Pour mettre à jour la répartition des voisins, il faut avoir connaissance des changements topologiques. GRABUP permet de prendre connaissance de certains de ces changements : en effet, lors de la réception d'un paquet *rep*, les nœuds savent qu'un de leurs voisins change de gradient. Comme il indique l'ancien h et le nouveau h', le nœud peut donc changer la répartition de ses voisins en conséquence (retirer un voisin à h et en ajouter un à h'). On note cependant que ce mécanisme ne permet de détecter des changements que pour des voisins encore en activité (on remarque tout de même que pour les nœuds qui meurent à cause d'un épuisement de la batterie, il est possible de leurs faire envoyer un paquet de mise à jour, paquet *rep*, quand celle-ci atteint un niveau critique). Le nœud, à chaque changement dans la répartition de ses voisins, recalcule son *offset* suivant la procédure de la section 4.1.

4.4 Conclusion

Dans ce chapitre, nous proposons un système de coordonnées virtuelles qui permet, au niveau MAC, de discriminer les nœuds et donc d'avoir des accès déterministes au médium. Au niveau routage, il donne le nombre de sauts pour atteindre le puits et permet un choix déterministe du prochain saut. De plus, ce SCV est construit en prenant en compte la connectivité de chaque nœud avec des voisins plus proches du puits, à distance égale ou plus éloignés du puits. Cette information peut être prise en compte pour améliorer le routage. On montre que notre proposition ne discrimine pas les nœuds dans un 2-voisinage de manière absolue. Cependant, les résultats théoriques et de simulation montrent que le nombre de collisions reste relativement faible, surtout quand le degré du réseau augmente. Notre proposition permet donc de discriminer de manière satisfaisante les nœuds dans un 2-voisinage. Les RCsF sont des réseaux dynamiques : des disparitions et apparitions de nœuds et de liens font constamment évoluer la topologie. Il est nécessaire que la coordonnée reflète l'état actuel du réseau pour le bon fonctionnement des protocoles. Nous proposons donc une méthode de mise à jour de gradient qui permet de mettre à jour seulement les parties du réseau touchées par des changements topologiques. Nous montrons par simulation que cette solution offre un meilleur compromis consommation énergétique/taux de livraison que le rafraîchissement périodique de tout le réseau. De plus, cette méthode de mise à jour de gradient permet aussi de mettre à jour la répartition des voisins d'un nœud et donc la coordonnée virtuelle proposée. Dans le chapitre suivant, nous propo-

sons un protocole de communication temps-réel localisé pour RCsF qui tire parti de l'organisation du réseau fournie par le SCV proposé dans ce chapitre pour effectuer des accès au canal déterministes et router les paquets sur des chemins de longueur bornée.

Chapitre 5

Proposition d'une solution de communication temps-réel

Dans le Chapitre 2, différentes solutions d'acheminement de données dans les RCsF sont étudiées. Aucune n'est complètement satisfaisante pour les applications critiques qui s'exécutent dans les RCsF. En effet, une partie des solutions proposées ne permet pas de répondre aux exigences des applications critiques en termes d'assurances de délais de bout en bout et de fiabilité des communications. Une autre partie des solutions (notamment les ordonnancements globaux) donnent des garanties sur les délais de bout en bout mais ne permettent pas de satisfaire les contraintes de consommation d'énergie et de passage à l'échelle inhérentes aux RCsF.

Nous concluons qu'une solution satisfaisante doit pouvoir :

- prioriser et borner les accès au médium des différents nœuds d'un domaine d'interférence pour introduire du déterminisme et ainsi fournir des garanties sur le délai d'accès au canal ;
- borner et connaître le nombre de sauts maximal dans le réseau pour borner le délai de bout en bout ;
- permettre un accès dynamique au canal, en laissant les nœuds qui ont du trafic y accéder, tandis que les autres restent endormis pour économiser l'énergie et ainsi augmenter la durée de vie du réseau ;
- les décisions de routage doivent se faire de manière locales pour que le protocole puisse passer l'échelle ;
- maximiser la fiabilité des communications de bout en bout pour obtenir un taux de livraison suffisant pour les applications critiques ;
- prendre en compte la consommation d'énergie pour augmenter la durée de vie du réseau.

Dans ce chapitre nous proposons RTXP (*Real-Time X-layer Protocol*), un protocole qui possède toutes ces caractéristiques. Ce protocole s'appuie sur le système de coordonnées virtuelles présenté dans le chapitre précédent pour organiser les accès au médium et sélectionner le prochain saut d'un paquet de manière déterministe. De plus, pour augmenter la fiabilité des communications de bout en bout, RTXP implémente un routage de type opportuniste. Ce type de routage est, comme montré dans le Chapitre 3, plus fiable que le routage classique.

Dans les sections suivantes, nous discutons les hypothèses prises pour la conception de RTXP (section 5.1), nous présentons l'idée générale du protocole (section 5.2), nous discutons le type de métriques utilisables pour que RTXP fonctionne (section 5.3). Nous détaillons ensuite notre proposition (section 5.5). Nous calculons les paramètres théoriques du protocole (délai et capacité) (section 5.6). Enfin, nous présentons ses performances et les comparons à celles de solutions de l'état de l'art (section 5.7).

5.1 Hypothèses

Dans cette section, nous décrivons les hypothèses considérées pour la conception de RTXP. Ces hypothèses sont principalement liées aux capacités des nœuds, à l'environnement radio et à l'application.

Hypothèses liées au nœuds :

- Les nœuds ont une source d'énergie limitée.
- Ils ont une mémoire limitée.

Hypothèses liées à l'environnement radio :

- Les radios des nœuds sont *half-duplex*.
- Ils ne peuvent communiquer que sur un seul canal.
- Nous considérons un modèle d'interférence à deux sauts. Cela signifie que les nœuds interférants (définition 2.1.1 p.14) d'un nœud sont ceux qui se trouvent dans son 2-voisinage (définition 2.1.3 p.14).

Hypothèse liée à l'application :

- Le trafic est faible (on parle de détection d'anomalie, donc d'événements qui n'arrivent pas fréquemment) et est composé exclusivement d'alarmes qui remontent vers le puits.

Hypothèse spécifique a notre proposition :

- Les nœuds sont munis d'une métrique qui leur donne le nombre de sauts qui les séparent du puits et qui est unique dans un 2-voisinage (dans ce chapitre nous utilisons les coordonnées issues du chapitre précédent).

5.2 Fonctionnement général

Pour limiter la consommation d'énergie, RTXP implémente un mécanisme de cycle d'endormissement. Nous appelons "période d'éveil", la période durant laquelle les nœuds ont une activité radio et "période de sommeil", la période durant laquelle les nœuds ont leur radio éteinte. RTXP utilise des paquets appelés *jamming codes*, ces paquets ne comportent pas d'information utile, ils sont composés d'une suite aléatoire de bits. Ces paquets peuvent être détectés dans le 2-voisinage de l'émetteur (modèle d'interférence à deux sauts).

Comme représenté sur la Figure 5.1, la période d'éveil est divisée en quatre parties :

- Une période de *backoff* durant laquelle les nœuds ayant un paquet dans leur *buffer* d'envoi, entrent en contention pour réserver le canal. Chaque nœud a un temps de *backoff* calculé à partir de sa métrique qui est unique dans un

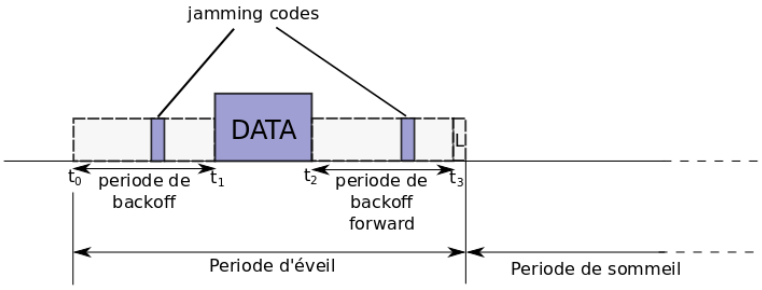


FIGURE 5.1 – Fonctionnement général de RTXP

2-voisinage. Pendant la période de *backoff*, les nœuds scrutent le canal. Soit le *backoff* expire sans que le nœud n'ait détecté de *jamming code* sur le canal, auquel cas il émet un *jamming code*. Soit il détecte un *jamming code* avant l'expiration de son *backoff* ce qui signifie qu'il a perdu l'accès au canal. Le nœud qui émet est celui dont la métrique est la plus petite.

- Un *slot* de temps pour transmettre le paquet de données (utilisé par le gagnant du *backoff* précédent). Le paquet est émis en *broadcast* entre t_1 et t_2.
- Une autre période de *backoff*, similaire à la première, durant laquelle les nœuds qui ont reçu le paquet de données et qui ont un nombre de sauts les séparant du puits inférieur à l'émetteur entrent en contention pour sélectionner le relayeur du paquet. Cette fois encore, la durée du *backoff* est calculée en fonction de la métrique.
- Un slot L, de la durée d'un *jamming code*. Il permet à un nœud qui a un paquet à envoyer mais qui a perdu la contention pour accéder au canal, de maintenir ses voisins éveillés en envoyant un *jamming code* dans ce slot (plusieurs nœuds peuvent émettre simultanément, car ils ne transmettent pas de données explicites). Ainsi, une nouvelle période d'éveil est déclenchée.

Comme discuté dans la section 5.5, l'unicité de la métrique permet des accès au médium et une sélection du relayeur déterministes.

5.3 Discussion sur la métrique

RTXP nécessite l'utilisation d'une métrique qui a deux propriétés : elle doit être unique dans un 2-voisinage et donner le nombre de sauts qui séparent un nœud du puits. Dans le chapitre précédent, nous présentons un SCV qui à pour but d'avoir ces deux propriétés. Cependant, avec cette proposition, des collisions de coordonnées persistent même si nous montrons que leur nombre est faible et limité. Dans ce chapitre, nous utilisons les coordonnées présentées dans le Chapitre 4 avec l'hypothèse qu'aucune collision n'apparaît lors de l'initialisation.

En pratique, si dans les topologies auxquelles ont applique les coordonnées, des collisions apparaissent de manière fréquente, on peut les éliminer en concaténant à la partie décimale de la coordonnée, un identifiant unique dans tout le réseau (provenant du matériel ou inclus dans le logiciel avant déploiement).

De manière générale, n'importe quelle métrique qui est unique dans un 2-voisinage et qui indique le nombre de sauts pour atteindre le puits peut être utilisée avec RTXP. Dans la suite de ce chapitre nous utilisons le SCV présenté dans le chapitre précédant avec l'hypothèse qu'il n'y ait pas de collisions de coordonnées. En plus des propriétés d'unicité dans un 2-voisinage et de gradient, elle permet de sélectionner le nœud le plus proche du gradient inférieur (cf. affinement du gradient décrit dans la section 4.1 le chapitre précédent) lors du relayage des paquets .

5.4 Synchronisation des nœuds : échantillonnage de préambule ou synchronisation globale ?

Dans la section 5.2 la présentation du fonctionnement général de RTXP, nous supposons que les nœuds d'un 2-voisinage sont éveillés au début de la première période de *backoff*. Cet instant est noté t_0 sur la figure 5.1. Comme présenté dans le Chapitre 2, il y a deux façons de synchroniser les nœuds.

La première solution utilise des préambules longs [Polastre et al., 2004]. Avec cette technique, un nœud qui a un paquet à envoyer, envoie un préambule dont la durée est égale à la longueur du cycle d'endormissement comme présenté dans la section 2.1.1 du Chapitre 2. Dans le cas de RTXP, la fin du préambule doit correspondre avec t_0. Avec cette solution, il n'y a pas besoin de maintenir de synchronisation entre les nœuds. Cependant, l'émission du préambule avant chaque transmission consomme de l'énergie. En effet, quand une alarme est relayée vers le puits, chaque nœud relayeur doit émettre un préambule avant d'exécuter les 4 phases du protocole.

La seconde solution consiste à synchroniser les nœuds du réseau localement ou globalement, de façon à ce que des nœuds qui doivent communiquer aient une horloge commune. Ces nœuds connaissent alors l'instant t_0 auquel ils doivent se réveiller. Cette solution peut paraître coûteuse en énergie, mais des travaux récents [Lampin, 2013] montrent qu'avec une technique de synchronisation adéquate, une synchronisation des nœuds même globale, peut être plus efficace en énergie que l'échantillonnage de préambule (cela dépend de l'intensité du trafic). De plus avec une synchronisation globale, un paquet peut être relayé plusieurs fois durant un cycle d'endormissement (dans le cas de l'échantillonnage de préambule il faut synchroniser le voisinage à chaque saut). Pour ces raisons, nous préférons une synchronisation globale pour RTXP. Dans le reste de ce chapitre nous considérons que les nœuds du RCsF sont synchronisés [Elson and Estrin, 2003]. Nous pouvons cependant noter que quelle que soit la solution utilisée, le protocole reste temps-réel.

5.5 Détails de RTXP

D'abord nous détaillons les phases décrites dans la section 5.2.

Phase 1. Durant la première période de *backoff* (phase B), chaque nœud qui a un paquet à envoyer entre en contention avec les nœuds de son 2-voisinage pour accéder au canal. S'il détecte une transmission avant la fin de son temps de *backoff*,

cela signifie qu'il a perdu la contention et donc l'accès au médium (dans ce cas, il envoie un *jamming code* dans le *slot* L. Les nœuds recevant ce *jamming code* restent éveillés pour une autre période). Sinon il émet un *jamming code* et peut envoyer son paquet dans la phase suivante. Comme mentionné précédemment, nous supposons que les nœuds sont capables de détecter les *jamming codes* émis dans leur 2-voisinage. Cela permet d'éviter le problème du terminal caché. La durée du *backoff* doit être unique dans un 2-voisinage pour éviter les collisions. Cette durée est calculée avec une fonction bijective qui prend comme paramètre la coordonnée (par exemple l'*offset* de la coordonnée est traduit directement en millisecondes, la fonction est donc du type $y = x$). On note que l'ordre d'accès des nœuds n'a pas d'importance, il faut seulement qu'il n'y ait pas de collision pour garantir des accès déterministes au médium. Le Lemme 5.5.1 assure que de cette façon il n'y a pas de collision dans un 2-voisinage.

Lemme 5.5.1 *Si les coordonnées sont uniques dans un 2-voisinage et que la fonction de* backoff *est bijective alors un seul nœud gagne la contention.*

Preuve Nous raisonnons par l'absurde. Supposons qu'au moins deux nœuds gagnent la contention (i.e. il y a une collision). Cela signifie qu'ils ont le même temps de *backoff*. Cela implique qu'ils aient la même coordonnée ou bien que la fonction de *backoff* donne le même temps pour plusieurs coordonnées donc n'est pas bijective. Ce sont des contradictions. ∎

.

Phase 2. Durant la deuxième phase (*réception* de données, notée phase R), le nœud (de gradient h) qui a gagné la contention de la phase 1 envoie son paquet. Les nœuds voisins le reçoivent et seuls les nœuds à $h - 1$ sauts du puits le conservent et passent à la phase 3.

Phase 3. La troisième phase est une autre période de *backoff* (*backoff forward* ou BF). Les nœuds qui reçoivent un paquet lors de la deuxième phase entrent en contention pour sélectionner lequel d'entre eux relaye le paquet. Comme pour la première phase, la fonction de *backoff* est bijective et calculée à partir de la coordonnée. Pour le choix du relayeur, nous voulons aussi préserver l'ordre donné par la coordonnée. En effet, nous voulons que les nœuds de gradient h ayant le plus de connections avec les nœuds de gradient $h - 1$ soient prioritaires pour relayer les paquets (on rappelle que cette information est donnée par l'*offset* de la coordonnée qui est un affinement du gradient). La fonction de *backoff* doit donc, en plus, être strictement monotone. Le premier nœud dont le *backoff* expire notifie aux autres qu'il est le relayeur en envoyant un *jamming code*. Ce mécanisme est une implémentation du routage opportuniste présenté dans le Chapitre 3.

Organisation des phases. Comme énoncé dans la section 5.4, nous choisissons d'utiliser une synchronisation globale des nœuds dans le réseau. Cela permet aux paquets d'être relayés plusieurs fois durant un cycle d'endormissement, car les relayeurs potentiels sont déjà synchronisés. Le modèle d'interférence à deux sauts permet à des nœuds éloignés de trois sauts de transmettre au même instant sans problème d'interférence. Pour donner une chance à chaque nœud du réseau de transmettre durant un

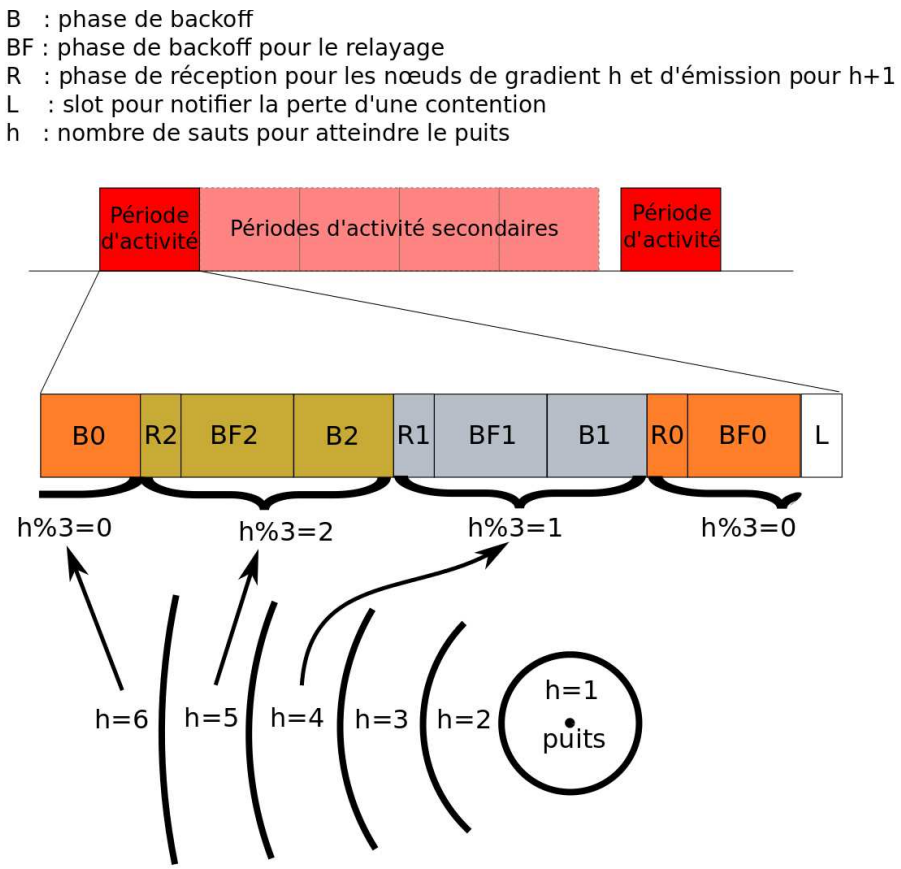


FIGURE 5.2 – Description détaillée de RTXP

cycle d'endormissement, quel que soit le nombre de sauts qui le sépare du puits, nous définissons 3 périodes d'éveil par cycle d'endormissement (la définition d'une période d'éveil est fournie dans la section 5.2). En effet, il faut une période d'éveil pour les nœuds à $3j$ sauts du puits, une pour ceux à $3j+1$ et une pour ceux à $3j+2$ avec $j \in \mathbb{N}$. Comme représenté par la figure 5.2, nous nommons "période d'activité" de RTXP la juxtaposition de trois périodes d'éveil. Un paquet peut, au plus, atteindre un nœud à $3j$ sauts du puits en partant d'un nœud à $3j+3$ sauts du puits durant une période d'activité. Pour que les paquets convergent vers le puits, il faut respecter l'ordre des périodes d'éveil de la Figure 5.2.

La figure 5.2 illustre les différentes phases pour des nœuds à différentes distances du puits en nombre de sauts. B_i, R_i et BF_i correspondent respectivement aux phases de *backoff*, *réception*, et *backoff forward*, avec $i = h \bmod 3$. Par exemple un nœud à 6 sauts du puits entre en contention pour le canal dans B_0 (phase 1) s'il a un paquet à envoyer. Il envoie le paquet dans R_2 (phase 2) s'il gagne la contention. Il se réveille dans R_0 (phase 2) pour éventuellement recevoir un paquet, et s'il en reçoit un, il exécute la phase 3 dans BF_0 pour tenter d'être le relayeur du paquet. Le reste du temps il dort.

Quand un nœud, qui a un paquet à transmettre, perd la contention pendant la phase de *backoff*, il demande une nouvelle période d'activité (appelée période d'acti-

vité secondaire) en envoyant un *jamming code* dans le *slot* L. Cette nouvelle période d'activité s'exécute à la suite de la première. Seuls les nœuds qui détectent un *jamming code* durant le *slot* L restent éveillés pour une période d'activité secondaire. On peut noter qu'à la fin d'une période d'activité secondaire, une autre peut être déclenchée de nouveau jusqu'à ce que tous les paquets aient avancés d'un saut vers le puits.

Dans les RCsF, les liens radios sont non-fiables (c.f. Chapitre 3), les nœuds sont soumis aux phénomènes de *shadowing* et de *fading*. Des paquets peuvent donc être perdus. Pour limiter l'impact des pertes de paquets sur la fiabilité, nous utilisons un routage opportuniste. Comme montré dans le Chapitre 3, ce type de routage permet d'obtenir une meilleure fiabilité que le routage classique pour lequel un paquet est adressé à un voisin en particulier. Dans le cas de RTXP, le nœud relayeur d'un paquet est sélectionné durant la phase BF. De plus, le *jamming code* envoyé durant la phase d'élection du relayeur (BF) est utilisé comme un acquittement implicite. En effet, un nœud qui envoie un paquet dans la phase R attend un *jamming code*. S'il ne le reçoit pas, le paquet est considéré comme perdu et le nœud émetteur envoie un *jamming code* dans le *slot* L pour demander une nouvelle période d'activité. Le paquet est réémis durant cette période.

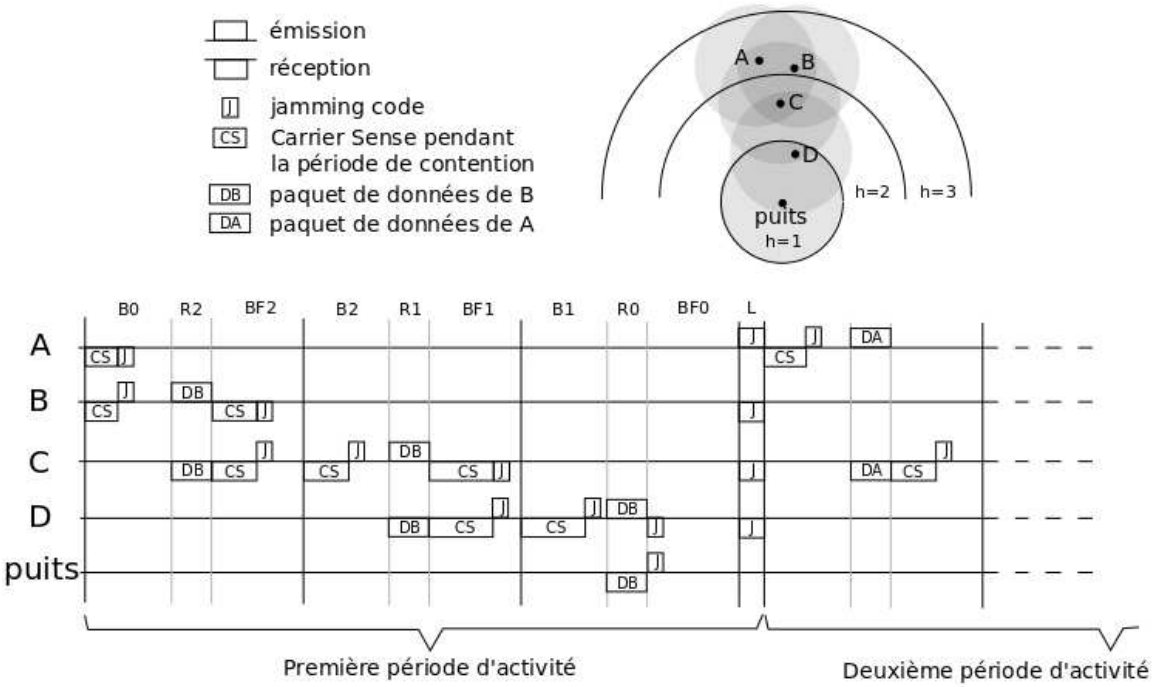


FIGURE 5.3 – Exemple avec 4 nœuds, les nœuds A et B ont un paquet à transmettre

Exemple. La Figure 5.3 représente un exemple d'exécution avec un réseau de 4 nœuds. Les nœuds A et B ont un paquet à envoyer. Ils entrent donc en contention dans B_0 (car $3 \mod 3 = 0$). B gagne la contention et envoie son paquet dans R_2, il est reçu par C. C est le seul récepteur, il est donc élu comme relayeur dans la phase BF_2. De la même manière que B, C envoie le paquet à D (en utilisant les phases B_2,

R_1 et BF_1) et D l'envoie au puits (B_1, R_0 et BF_0). A la fin de la période d'activité, le nœud A envoie un *jamming code* dans le *slot* L car il a perdu la contention B_0. B, C et D détectent un *jamming code*, ils restent donc éveillés pour une nouvelle période d'activité. Durant cette période d'activité secondaire, seul A a un paquet à envoyer. Il gagne donc la contention et le paquet est envoyé à C. A la fin de la seconde période d'activité, tous les nœuds s'endorment car aucun ne transmet durant le *slot* L (chaque paquet a fait au moins un saut en direction du puits).

Symbole	Signification
D_B	Durée d'une phase de *backoff* (B)
D_{BF}	Durée d'une phase *backoff forward* (BF)
D_R	Durée d'une phase de *reception* (R)
D_L	Durée du *slot* L
$D_{jamming}$	Durée du *jamming code*
$D_{activity\_period}$	Durée d'une période d'activité
D_{awake}	Durée de la période d'éveil d'un nœud pendant une période d'activité
D_{sleep}	Durée de la période de sommeil
$WCTT_{RTXP}$	Borne théorique sur le délai de bout en bout de RTXP
DC	Rapport de cycle d'endormissement : durée de la période d'éveil divisée par la durée du cycle d'endormissement
C_{RTXP}	Capacité de RTXP
NB_{hop_max}	Nombre maximum de sauts pour atteindre le puits
$E_{backoff}$	Energie consommée durant la phase de *backoff* (B)
$E_{backoff\_forward}$	Energie consommée durant la phase *backoff forward* BF
E_{TX_packet}	Energie consommée durant l'émission d'un paquet
E_{RX_packet}	Energie consommée durant la réception d'un paquet
E_{1hop_RTXP}	Energie consommée par RTXP pour réaliser un saut
k	Degré moyen du réseau

TABLE 5.1 – Notations utilisées pour la description de RTXP

5.6 Calcul des paramètres théoriques de RTXP

Dans cette section nous donnons les équations des paramètres de RTXP et évaluons les compromis entre le délai, la capacité et l'énergie.

5.6.1 Délai, capacité et énergie

Pour calculer une borne sur le délai de bout en bout (cas pire de délai de bout en bout ou *Worst Case Traversal Time*, WCTT), nous calculons le pire cas de délai pour un saut et nous multiplions cette valeur par le nombre de sauts maximum pour atteindre le puits. Pour commencer, nous définissons des délais intermédiaires. Les notations utilisées dans cette section sont détaillées dans le Tableau 5.1.

Les durées de B et BF dépendent du temps de *backoff* qui est fonction de l'*offset* ($backoff = f(offset)$). Tous les nœuds doivent avoir la possibilité d'envoyer un *jamming code* pendant la durée de la phase de *backoff*. La durée de cette phase doit donc être égale au temps de *backoff* maximal plus la durée d'un *jamming code* :

$$D_B = D_{BF} = max(backoff) + D_{jamming} \tag{5.1}$$

La durée de la phase R est simplement le temps nécessaire pour transmettre un paquet de données (noté D_R). Dans notre cas, un paquet d'alarme a une taille de l'ordre de quelques dizaines d'octets.

La durée du *slot* L (D_L) est égale à la durée d'un *jamming code* ($D_{jamming}$).

La durée d'une période de sommeil (D_{sleep}) est déduite de la durée d'une période d'éveil (D_{awake}). La durée de la période d'éveil d'un nœud est égale à la durée de la phase B, et de la phase BF et de deux phases R (une pour recevoir un paquet et une pour en envoyer un). Le calcul de D_{sleep} dépend du rapport du cycle d'endormissement (DC), qui dépend de l'application. Nous avons donc :

$$DC = D_{awake}/(D_{sleep} + D_{awake}) \tag{5.2}$$

$$D_{awake} = D_B + D_{BF} + 2 \times D_R + D_L \tag{5.3}$$

$$D_{sleep} = D_{awake} \times (\frac{1}{DC} - 1) \tag{5.4}$$

Le rapport du cycle d'endormissement dépend de l'application, cet aspect est discuté dans la section 5.6.2.

La période d'activité qui est détaillée sur la Figure 5.2 a une durée de :

$$D_{activity\_period} = 3 \times (D_B + D_{BF} + D_R) + D_L \tag{5.5}$$

Le pire cas pour la durée d'un saut correspond à $D_{activity\_period} + D_{sleep}$. En effet, un nœud qui a un paquet à transmettre doit gagner l'accès au canal dans son 2-voisinage (phase B). S'il y a plusieurs paquets dans le 2-voisinage, des périodes d'activité secondaires sont exécutées. Cependant, le dernier paquet doit être transmis avant l'instant de réveil des nœuds du réseau (noté t_0). Donc un paquet peut être retardé au plus d'une durée d'un cycle d'endormissement. Pour la borne sur le délai de bout en bout nous prenons le nombre de sauts maximal plus un, car il y a un délai entre la détection de l'événement et l'émission du paquet (ce délai correspond au plus au délai d'un cycle d'endormissement donc d'un saut).

$$WCTT_{RTXP} = (NB_{hop\_max} + 1) \times (D_{activity\_period} + D_{sleep}) \tag{5.6}$$

Le nombre de périodes d'activité est limité par la longueur de la période de sommeil (qui dépend du cycle d'endormissement). Nous définissons la capacité de RTXP (C_{RTXP}) comme étant le nombre de paquets qui peuvent être transmis dans un 2-voisinage pendant la durée d'un cycle d'endormissement. La durée du cycle d'endor-

missement étant $D_{activity\_period} + D_{sleep}$, la capacité est :

$$C_{RTXP} = \left\lfloor \frac{D_{activity\_period} + D_{sleep}}{D_{activity\_period}} \right\rfloor \tag{5.7}$$

car on peut envoyer un seul paquet d'un 2-voisinage de gradient h durant $D_{activity\_period}$ donc C_{RTXP} paquets durant un cycle d'endormissement.

Théorème 5.6.1 *Posons $h \in \mathbb{N}$ et $p \in \mathbb{N}$ respectivement le nombre de sauts séparant un nœud du puits et le nombre de paquets dans un 2-voisinage de gradient h. En supposant qu'un paquet ne peut être perdu qu'à cause d'une collision, tous les paquets de tous les 2-voisinages au gradient h atteignent $h-1$ en au plus un cycle d'endormissement si $p < C_{RTXP}$.*

Preuve Nous faisons une preuve par l'absurde. Supposons qu'un paquet n'atteigne pas le gradient $h-1$ depuis h en un cycle d'endormissement. Soit le paquet est perdu, soit il est retardé pendant plus de la durée d'un cycle d'endormissement. Nous supposons que la seule façon de perdre le paquet est qu'une collision survienne, or nous savons par le Lemme 5.5.1 que cela n'est pas possible, c'est donc une contradiction. Si le paquet est retardé jusqu'à la fin du cycle d'endormissement, cela signifie que le nœud perd toutes les contentions possibles durant ce cycle. Il perd donc plus de $\lfloor \frac{D_{activity\_period}+D_{sleep}}{D_{activity\_period}} \rfloor = C_{RTXP}$ contentions, cela signifie qu'il y a plus de C_{RTXP} paquets ($p > C_{RTXP}$) dans un 2-voisinage, ce qui est une contradiction. ∎

En-dessous de cette limite de capacité, le taux de livraison est donc de 100% si on fait l'hypothèse que les pertes de paquets sont seulement dues aux collisions. Comme mentionné précédemment, ce n'est pas le cas en pratique, car les nœuds sont soumis à des phénomènes de *fading*. Nous avons aussi mentionné que l'impact de ce problème est atténué par l'utilisation d'un routage opportuniste et de retransmissions. Cet aspect est discuté dans la section d'évaluations des performances (5.7).

L'énergie dépensée pour qu'un paquet avance d'un saut en direction du puits est :

$$\begin{aligned} E_{1hop\_RTXP} &= p \times E_{backoff} + E_{TX\_packet} \\ &\quad + k \times E_{RX\_packet} + k \times E_{backoff\_forward} \end{aligned} \tag{5.8}$$

avec k le nombre de voisins de l'émetteur et p le nombre de paquets émis par des nœuds plus prioritaires. On constate que la consommation d'énergie dépend du degré du réseau. Avec le routage opportuniste, plus le réseau est dense, plus les communications sont fiables (c.f. Chapitre 3) mais plus le coût énergétique est important.

5.6.2 Compromis délai/capacité

La capacité dépend de l'inverse du cycle d'endormissement (c.f. équations 5.2 et 5.7). Plus le cycle d'endormissement est long, plus la capacité est importante. Cependant, la borne sur le délai de bout en bout augmente avec la durée du cycle

d'endormissement (équation 5.6). Dans le cas d'une application avec un trafic faible et de fortes contraintes temporelles (échéances courtes), on préférera une période de sommeil courte. Dans le cas d'une application avec un trafic plus important et des contraintes temporelles plus larges, une période de sommeil plus importante peut être utilisée.

Paramètre	Valeur
Nombre maximal de sauts pour atteindre le puits	5
Durée d'un *jamming code*	200μs
Durées des phases B et BF	10.2ms
Durée de la phase R	32ms
Rapport de cycle d'endormissement	de 100% à 1%

TABLE 5.2 – Paramètres utilisés pour tracer la courbe de capacité en fonction du WCTT

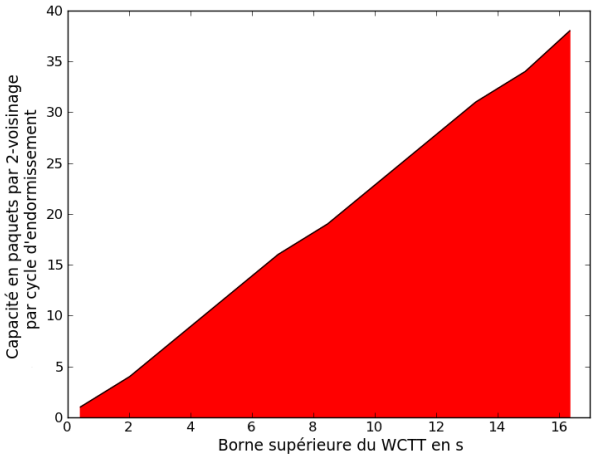


FIGURE 5.4 – Capacité de RTXP en fonction du WCTT

La Figure 5.4 illustre la capacité en nombre de paquets par 2-voisinage qui peuvent être traités pendant un cycle d'endormissement en fonction du pire délai de bout en bout (ce délai dépendant lui-même du cycle d'endormissement). La partie sous la courbe correspond à une zone de faisabilité pour RTXP, c'est-à-dire que dans cette zone, le trafic peut être écoulé tout en respectant la borne temporelle. Par exemple, avec les valeurs présentées dans le Tableau 5.2, si l'application requiert une borne de 6 secondes, la capacité maximale de RTXP est égale à 15. Cela signifie que 15 paquets peuvent être transmis dans un 2-voisinage durant un cycle d'endormissement.

5.7 Performances

Dans cette section, nous évaluons par simulation les performances de la solution proposée et nous les comparons à deux protocoles de la littérature. D'abord,

nous comparons RTXP avec une solution temps-réel centralisée, PEDAMACS [Ergen and Varaiya, 2006]. Cependant, avec des liens non-fiables, il n'est pas possible de donner de garanties strictes sur les délais de bout en bout. En effet, il n'est pas possible de connaître avec certitude le nombre de transmissions nécessaires pour qu'un paquet soit effectivement reçu. Nous comparons donc RTXP avec une solution non déterministe, pour montrer qu'une solution déterministe reste plus fiable même avec des conditions radios hostiles.

5.7.1 Environnement et paramètres de simulation

Comme pour le Chapitre 4, les simulations ont été réalisées avec le simulateur à événements discrets WSNet [WSNet, 2009]. Pour la campagne de simulations, nous avons généré 140 topologies aléatoires, pour lesquelles les nœuds sont distribués uniformément sur un plan de 50×50 unités de longueur. Ces topologies sont divisées en ensembles de 20 topologies de $m \times 100$ nœuds avec m un entier $\in [2, 8]$. Une simulation est réalisée sur chaque topologie.

Paramètre	valeur
Nombre de nœuds	100 à 800
Débit	500kbps
Porté radio	10 unités
Aire	50×50 unités
Taille d'un paquet	100 octets
Durée d'un *jamming code*	200 μs
Durée des phases B et BF	10,2 ms
Rapport du cycle d'endormissement	1%
Exposant des pertes en espace	2
σ de la loi log-normale	4

TABLE 5.3 – Paramètres de simulation

Pendant chaque simulation, 200 paquets sont envoyés. Le trafic est composé d'alarmes générées de façon périodique depuis un point choisi de manière aléatoire dans le réseau : un nœud est choisi aléatoirement parmi les nœuds du réseau pour être la source de l'alarme. Dans les simulations, nous considérons deux débits d'alarmes, une alarme toutes les 5 secondes et une alarme toutes les secondes. De cette façon, nous observons le comportement de RTXP soumis à différentes charges. Pour les topologies simulées, ces charges de trafic sont très inférieures à la limite de capacité de RTXP exprimée par l'Equation 5.7. En effet, le rapport du cycle d'endormissement est de 1% : des Equations 5.4 et 5.5 nous pouvons déduire que la longueur d'un cycle est d'environ 2,5 secondes. D'après l'équation 5.7, cela signifie qu'au plus 100 paquets peuvent être relayés durant un cycle dans un 2-voisinage (environs 40 paquets par seconde). Avec une alarme toutes les 5 secondes et une alarme toutes les secondes, nous nous plaçons dans des cas qui correspondent à l'hypothèse de faible trafic énoncé dans la section 5.1. Cela permet aux nœuds de dormir la plupart du temps pour économiser l'énergie (peu de périodes d'activité secondaires sont déclenchées).

Nous utilisons un modèle de transmetteur mono-canal *half-duplex* avec un débit de 500kbps. Les paramètres de simulations sont détaillés dans le Tableau 5.3.

Deux modèles de propagation sont utilisés : le modèle de pertes en espace libre et le modèle *log-normal shadowing*. Le modèle de perte en espace libre nous permet d'évaluer les performances de RTXP dans le cas où les pertes de paquets sont seulement dues aux interférences entre nœuds. Cela nous permet de confronter les affirmations établies dans la section 5.6 aux résultats de simulations. Le modèle *log-normal shadowing* permet d'évaluer les performances du protocole dans des conditions plus réalistes (c.f. Chapitre 3).

5.7.2 Comparaison avec PEDAMACS

Dans cette section nous présentons les résultats de simulation de RTXP et PEDAMACS avec les modèles de pertes en espace libre et *log-normal shadowing*. Nous comparons les délais de bout en bout, l'énergie consommée lors des simulations et le taux de livraison des deux protocoles.

Paramètres théoriques de PEDAMACS

A notre connaissance, RTXP est le premier protocole qui est à la fois, temps-réel, *cross-layer* et localisé pour les RCsF. Les solutions *cross-layers* temps-réel existantes sont centralisées. Nous nous comparons donc à ce type de solutions. Comme décrit dans la section 2.2.3 du Chapitre 2, dans le cas de PEDAMACS, le puits produit une trame d'ordonnancement globale basée sur une connaissance globale de la topologie du réseau (acquise au préalable lors d'une phase d'initialisation). Dans cette section, nous définissons le délai pire cas de PEDAMACS ainsi que la consommation d'énergie nécessaire pour qu'un paquet fasse un saut en direction du puits.

Dans [Coleri, 2002], les auteurs affirment que tous les paquets atteignent le puits durant au plus la durée d'une phase d'ordonnancement (c'est-à-dire avant la fin de la trame d'ordonnancement). La longueur maximale de la trame d'ordonnancement dépend de la topologie du réseau (représenté par un arbre enraciné au puits), plusieurs cas sont détaillés dans [Coleri, 2002]. Dans ce chapitre, nous considérons le cas d'un arbre $G = (V, E)$ avec un modèle d'interférence à 2 sauts. Ce graphe est récupéré par le puits durant les phases d'initialisation (c.f. section 2.2.3). Dans ce cas, la durée maximale de la trame est :

$$WCTT_{PEDAMACS} = 3 \times (|V| - 1) \times T_{slot} \tag{5.9}$$

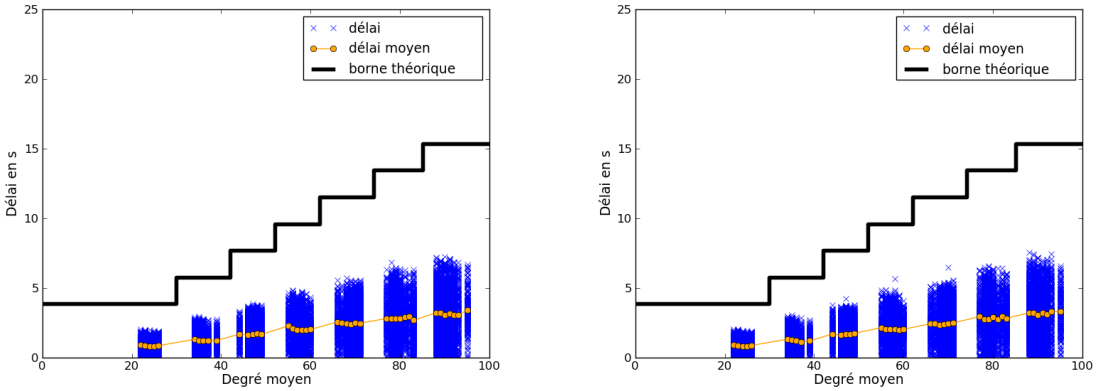
L'Equation 5.9 montre que, dans le cas de PEDAMACS, la borne sur le délai de bout en bout ne dépend pas directement du nombre de sauts entre la source et la destination mais du nombre de nœuds (le pire cas étant un réseau ligne, le nombre de nœuds $|V| - 1$ est alors égal au nombre de sauts).

Nous évaluons l'énergie consommée pour qu'un paquet effectue un saut en direction du puits. Pour PEDAMACS, cela correspond seulement à l'énergie nécessaire pour envoyer le paquet et pour le recevoir (on ne prend pas en compte l'initialisation).

$$E_{1hop\_PEDAMACS} = E_{TX\_packet} + E_{RX\_packet} \tag{5.10}$$

Lors des simulations, nous comparons les délais observés avec les bornes théoriques de RTXP et PEDAMACS, exprimées respectivement par les Equations 5.6 et 5.9. Sur les graphes présentés dans cette section, nous avons choisi de représenter le délai de bout en bout de chaque paquet (croix sur les graphes) pour pouvoir observer la distribution des valeurs. Nous représentons également le délai moyen (courbe avec les cercles) et la borne théorique (trait épais).

Modèle de pertes en espace libre.



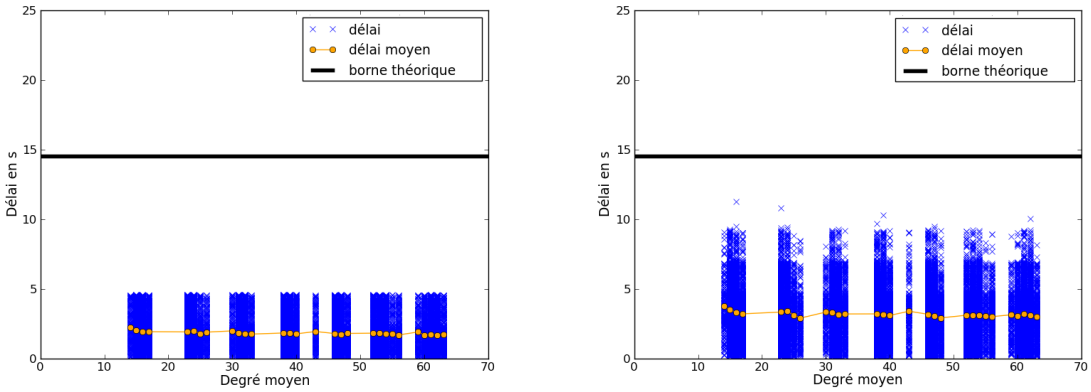
(a) 1 paquet toutes les 5 secondes

(b) 1 paquet par seconde

FIGURE 5.5 – Délais de PEDAMACS (cas des pertes en espace libre)

Les Figures 5.5(a) et 5.5(b) représentent le délai de bout en bout des alarmes en fonction du nombre moyen de voisins dans le réseau pour PEDAMACS, respectivement pour des trafics de 1 paquet toutes les 5 secondes et 1 paquet par seconde. Nous notons d'abord que tous les paquets respectent leurs échéances. Cela est assuré par l'ordonnancement global. De plus, l'ordonnancement assure aussi que deux nœuds interférants ne peuvent pas communiquer au même instant. Comme c'est la seule source de perte de paquets, nous observons un taux de livraison de 100%. Le délai n'est pas affecté par la charge de trafic car la trame d'ordonnancement ne change pas en fonction du trafic.

Les Figures 5.6(a) et 5.6(b) représentent le délais de bout en bout des alarmes et la borne théorique pour RTXP. Dans ce cas aussi nous observons que tous les paquets sont reçus avant l'échéance, comme prédit dans la section 5.6. Le taux de livraison est de 100%. Par contre dans le cas de RTXP, l'augmentation de la charge de trafic affecte le délai, cela produit une augmentation du délai de certains paquets. Cela est dû au fait que lorsque la charge augmente, cela déclenche plus de périodes d'activité secondaires car il y a plus de paquets au même instant dans un même 2-voisinage. En revanche, le délai ne varie pas en fonction du degré moyen du réseau.



(a) 1 paquet toutes les 5 secondes

(b) 1 paquet par seconde

FIGURE 5.6 – Délais de RTXP (cas des pertes en espace libre)

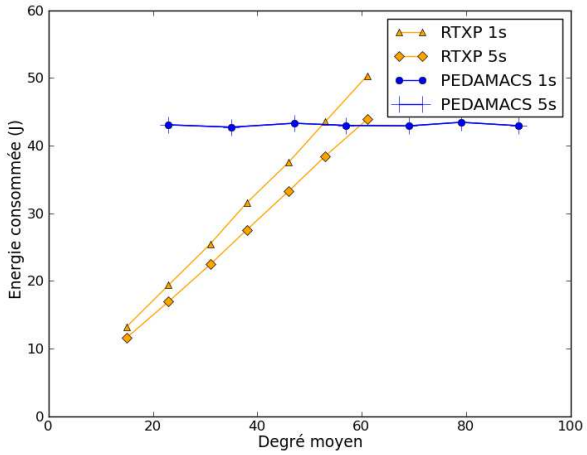


FIGURE 5.7 – Consommation d'énergie maximale (sur 20 simulations par point) de PEDAMACS et RTXP

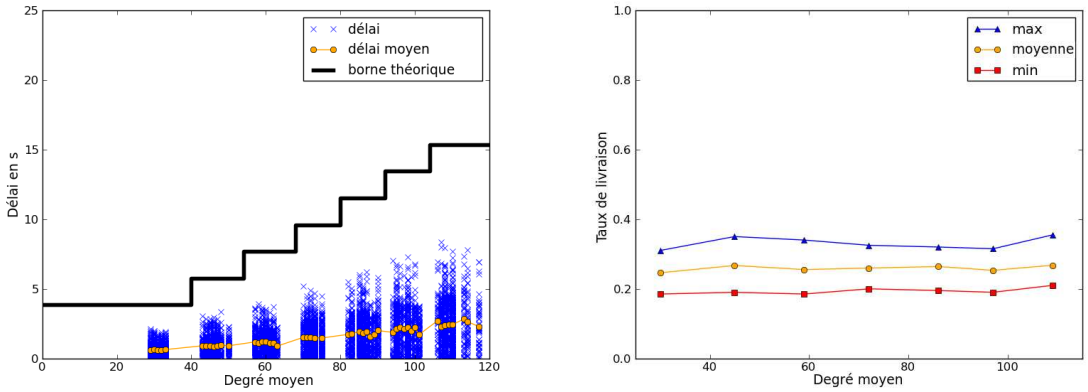
Consommation d'énergie. La Figure 5.7 représente la consommation d'énergie maximale observée. Chaque point de la courbe est le maximum de 20 simulations sur des topologies comportant le même nombre de nœuds. Le calcul d'énergie pour RTXP et PEDAMACS se fait respectivement d'après les Equations 5.8 et 5.10.

L'énergie consommée par PEDAMACS n'augmente pratiquement pas quand le degré moyen du réseau augmente. L'énergie dépensée est fonction de l'ordonnancement et du trafic, car les nœuds se réveillent dans les *slots* qui leurs sont alloués pour recevoir des paquets (même s'il n'y a pas de trafic) et envoient dans les *slots* d'émission (seulement s'ils ont des paquets). Le nombre de paquets émis étant toujours le même dans les simulations, l'augmentation de la charge n'augmente pas la consommation d'énergie.

L'énergie dépensée par RTXP augmente linéairement avec le degré moyen. Comme on l'a évoqué dans la section 5.6, c'est dû au fait que les paquets sont envoyés en *broadcast* aux voisins de l'émetteur. L'augmentation de la charge de trafic provoque une augmentation de la consommation d'énergie, c'est dû au déclenchement de plus de périodes d'activité secondaires.

PEDAMACS a une plus forte consommation d'énergie que RTXP pour des réseaux avec un degré moyen en dessous de 50. Cela est dû au fait qu'avec PEDAMACS les nœuds se réveillent, même s'il n'y a pas de trafic, à cause de l'ordonnancement. PEDAMACS est donc plus adapté à un trafic périodique où tous les nœuds ont un paquet à envoyer dans chaque trame d'ordonnancement, plutôt qu'à des remontées d'alarmes (pour lesquelles le trafic est moins important et apériodique). RTXP au contraire s'adapte au trafic. S'il n'y a pas d'alarme, les nœuds dorment la plupart du temps, s'il y a beaucoup d'alarmes, des périodes d'activité secondaires sont déclenchées pour écouler le trafic.

Modèle de propagation *Log-normal shadowing*

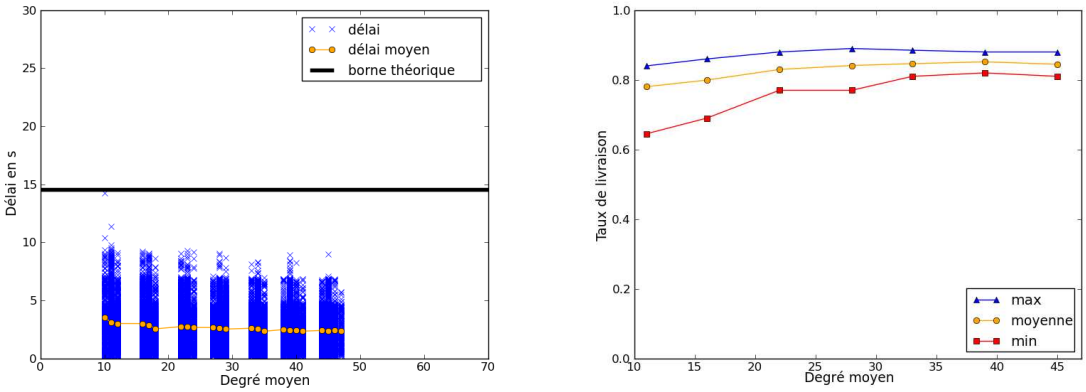


(a) Délai (b) Taux de livraison

FIGURE 5.8 – Délais et taux de livraison de PEDAMACS (cas du modèle *log-normal shadowing*)

La Figure 5.8(a) représente le délai de bout en bout et la borne théorique pour PEDAMACS dans le cas du modèle de propagation *log-normal shadowing*. Comme dans le cas du modèle de pertes en espace libre, on observe qu'aucun paquet n'arrive au puits après l'échéance. Cependant, la Figure 5.8(b) représente les taux de livraison minimum, maximum et moyen observés durant les simulations. Les valeurs sont faibles (entre 20 et 30 % de paquets reçus). La plupart des paquets sont perdus à cause des mauvaises conditions du canal radio. De plus, PEDAMACS n'implémente pas de mécanisme de retransmission de paquet, quand une transmission échoue, le paquet est définitivement perdu.

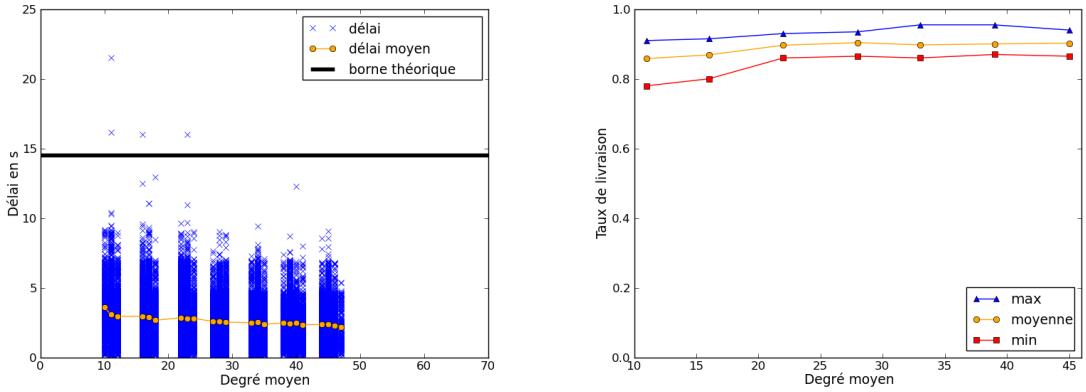
La figure 5.9(a) représente le délai de bout en bout et la borne théorique pour RTXP dans le cas du modèle de propagation *log-normal shadowing* sans retransmis-



(a) Délai (b) Taux de livraison

FIGURE 5.9 – Délais et taux de livraison de RTXP (cas du modèle *log-normal shadowing* sans retransmission)

sion. Comme dans le cas de PEDAMACS, tous les paquets qui atteignent le puits respectent l'échéance. Mais dans ce cas, le taux de livraison des paquets (Figure 5.9(b)) est plus important. C'est grâce au routage opportuniste qui augmente la fiabilité comme montré théoriquement dans le Chapitre 3.



(a) Délai (b) Taux de livraison

FIGURE 5.10 – Délais et taux de livraison de RTXP (cas du modèle *log-normal shadowing* avec retransmissions)

La figure 5.9(a) représente le délai de bout en bout et la borne théorique pour RTXP dans le cas du modèle de propagation *log-normal shadowing* avec retransmissions. Dans ce cas, on remarque que quelques paquets ne respectent pas l'échéance. Le mécanisme de retransmission est décrit dans la section 5.5, si le nœud émetteur d'un paquet ne reçoit pas de *jamming code* durant la phase BF, il envoie un *jamming code* dans le *slot* L et cela déclenche une nouvelle période d'activité durant laquelle

l'émetteur retransmet le paquet. Pour limiter la consommation d'énergie, un nœud peut tenter de transmettre un même paquet 5 fois par cycle d'endormissement (il pourra ensuite retenter dans le cycle suivant). Dans certains cas, le paquet peut être retardé de plusieurs cycles d'endormissement, il rate donc l'échéance. Cependant, le mécanisme de retransmission permet d'obtenir un meilleur taux de livraison même avec de mauvaises conditions du canal radio. La Figure 5.10(b) représente les taux de livraisons minimum, maximum et moyen observés durant les simulations (les délais restent inchangés et ne sont donc pas représentés ici). Les valeurs sont plus élevées que dans les cas de PEDAMACS et de RTXP sans retransmission.

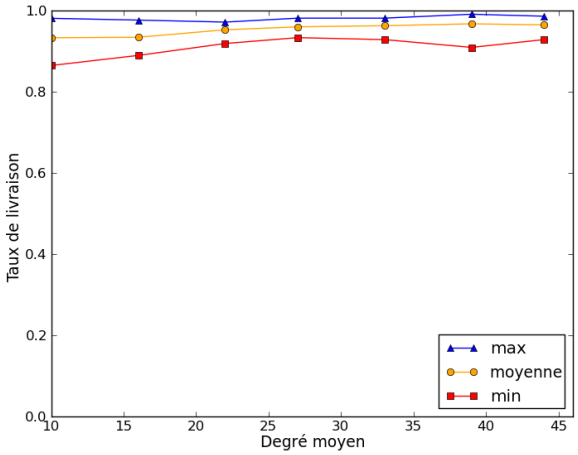


FIGURE 5.11 – Taux de livraison de RTXP (cas du modèle *log-normal shadowing* avec retransmissions et deux puits)

Dans le Chapitre 3 nous concluons que dans le cas d'un routage opportuniste, l'utilisation de plusieurs puits augmente la fiabilité. Nous observons donc l'impact de l'ajout d'un deuxième puits à côté du premier sur le taux de livraison de RTXP. La Figure 5.11 représente les taux de livraison minimum, maximum et moyen pour RTXP avec retransmissions et deux puits. On constate en effet une augmentation significative du taux de livraison, confirmant les résultats théoriques du Chapitre 3.

Avec des liens non-fiables, il est impossible de garantir qu'absolument tous les paquets sont reçus. Il n'est pas non plus possible de garantir qu'ils sont tous reçus avant l'échéance. Cela est dû à la nature probabiliste du lien radio. En effet, il existe une probabilité non nulle pour qu'un paquet ne soit pas reçu même après de nombreuses retransmissions. RTXP est conçu pour éviter les comportements probabilistes, l'accès au canal et la sélection du relayeur sont déterministes. Le comportement du protocole est prédictible et permet d'assurer que les paquets respectent leurs échéances. Cependant, le lien radio introduit un comportement probabiliste, on peut donc légitimement se demander dans quelle mesure il est utile d'avoir un protocole déterministe avec un canal probabiliste. Cet aspect est exploré dans la section suivante.

5.7.3 Comparaison avec une solution non temps-réel

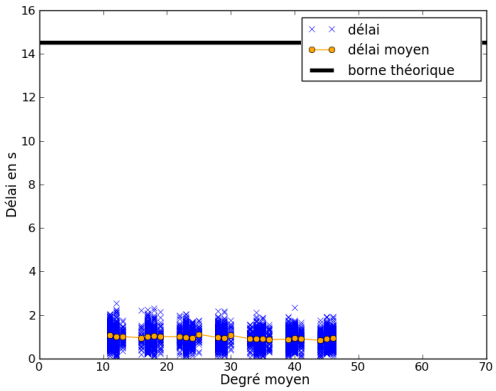
Dans cette section nous comparons RTXP avec une solution non temps-réel, cela nous permet de confirmer que, même avec un lien radio non-fiable, un protocole déterministe a de meilleures performances temps-réel. Nous choisissons de comparer RTXP à une solution composée de X-MAC au niveau MAC et un routage par gradient. X-MAC est une couche MAC asynchrone à échantillonnage de préambule présentée dans la section 2.1.1 du Chapitre 2. Son préambule est composé de paquets courts contenant l'adresse du nœud destination. Si celui-ci se réveille il peut répondre immédiatement, ce qui stoppe le préambule et déclenche l'envoie du paquet. Dans notre cas, nous utilisons un routage par gradient opportuniste. C'est-à-dire que n'importe quel nœud qui reçoit le paquet de préambule et qui est plus proche du puits en nombre de sauts peut répondre et donc devenir relayeur. Un nœud qui a un paquet à envoyer utilise la technique de CSMA/CA présentée dans le Chapitre 2, avant d'émettre il scrute le canal, s'il détecte de l'activité radio, il patiente pour une durée aléatoire avant de retenter. L'accès au canal est donc non-déterministe.

Avec cette solution, le délai de bout en bout dépend du nombre de sauts qu'un paquet doit réaliser pour atteindre le puits. Un paquet doit attendre au maximum un cycle d'endormissement avant de pouvoir faire un saut (la taille d'un préambule). Pour comparer de manière équitable RTXP et X-MAC avec routage par gradient, nous configurons la longueur du cycle d'endormissement comme étant un tiers de celle de RTXP (car avec RTXP un paquet peut faire au plus 3 sauts par cycle). Cette configuration désavantage en fait RTXP car un préambule de X-MAC dure en moyenne la moitié de la durée d'un cycle (temps moyen pour qu'un nœud réponde). Nous utilisons l'équation 5.6 comme borne théorique pour X-MAC avec gradient. X-MAC utilise un mécanisme d'acquittement et de retransmissions. Durant les simulations, différentes limites de nombres de retransmissions sont testées pour observer l'effet des retransmissions sur le délai et la fiabilité.

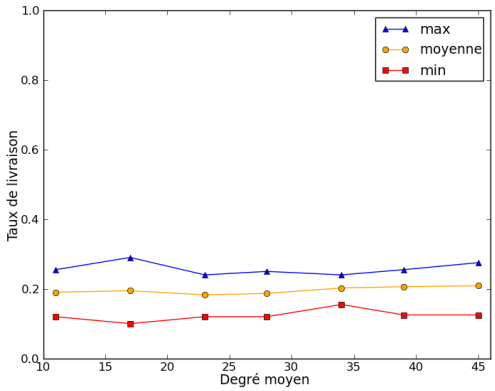
Pour les résultats de cette section, les simulations ont été réalisées avec un trafic de 1 alarme toutes les 5 secondes. Le modèle de canal utilisé est le *log-normal shadowing*. Les Figures 5.12, 5.13 et 5.14 décrivent respectivement les résultats pour 0, 5 et 500 retransmissions. Les paramètres observés sont le délai de bout en bout et le taux de livraison.

Sur la Figure 5.12(a) on constate que tous les paquets qui arrivent au puits respectent l'échéance dans le cas sans retransmission. Cependant, le taux de livraison, représenté sur la Figure 5.12(b), est très bas en comparaison de celui de RTXP (c.f. Figure 5.9(b)).

Dans le cas de la configuration à 5 retransmissions maximum, on observe sur la Figure 5.13(a) que quelques paquets ne respectent pas l'échéance. C'est dû au fait que les retransmissions augmentent le délai de bout en bout. Le taux de livraison, décrit par la Figure 5.13(b) est plus grand que dans le cas précédent. Cependant, le nombre de paquets qui ne respectent pas l'échéance est plus important que dans le cas de RTXP comme on peut le constater en comparant les Figures 5.13(a) et 5.10(a). Le taux de livraison est plus important que dans le cas précédent, car les retransmissions fiabilisent les échanges de paquets. On peut aussi noter que la différence entre les

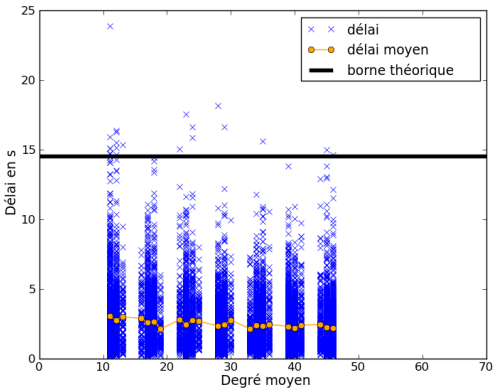


(a) Délai

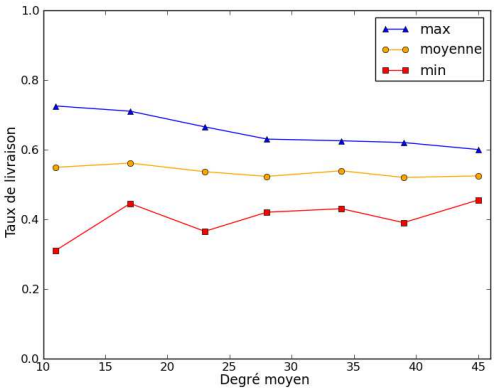


(b) Taux de livraison

FIGURE 5.12 – Délais et taux de livraison de X-MAC avec gradient : sans retransmission



(a) Délai

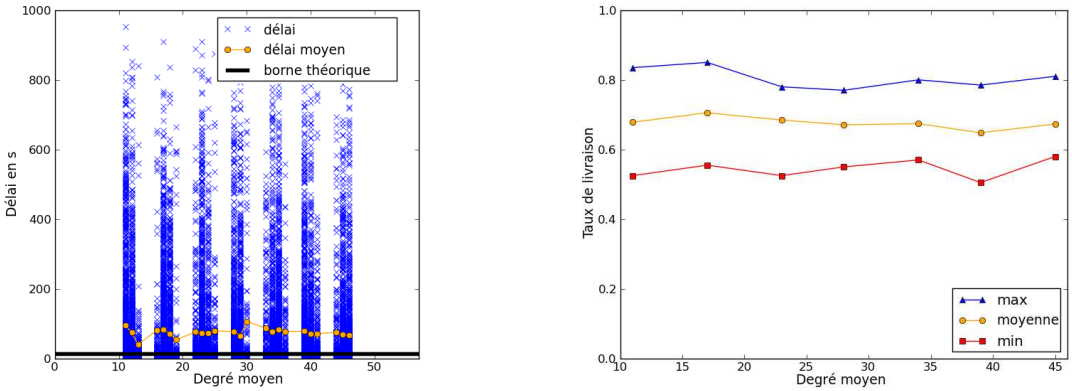


(b) Taux de livraison

FIGURE 5.13 – Délais et taux de livraison de X-MAC avec gradient : 5 retransmissions maximum

taux de livraisons maximum et minimum est plus petite dans le cas de RTXP, on peut donc en déduire qu'il a un comportement plus stable que X-MAC avec gradient pour différentes topologies.

Dans notre implémentation de RTXP, un paquet est retransmis 5 fois, au maximum dans un cycle d'endormissement. S'il n'est toujours pas reçu, il peut être retransmis au cycle suivant. Cela signifie qu'il n'y a pas de borne sur le nombre de retransmissions. Pour comparer équitablement RTXP et X-MAC avec gradient, nous choisissons un nombre très important de retransmissions : 500. Comme illustré par la Figure 5.14(a), la plupart des paquets ne respectent pas l'échéance, certains ont des délais de plusieurs centaines de secondes. On observe sur la Figure 5.14(b) que le taux de livraison est modérément augmenté par rapport au cas où il y a 5 retransmissions.



(a) Délai

(b) Taux de livraison

FIGURE 5.14 – Délais et taux de livraison de X-MAC avec gradient : 500 retransmissions maximum

Le taux de livraison reste cependant inférieur à celui de RTXP (c.f. Figure 5.10(b)). Les délais importants viennent du fait que le nombre important de retransmissions entraînent une forte occupation du canal ce qui induit des longs délais pour accéder au canal.

Ces résultats montrent que l'introduction de mécanismes déterministes pour l'accès au canal et le choix du relayeur permet de meilleures performances, même avec un lien radio non-fiable.

5.8 Conclusion

Dans ce chapitre nous présentons RTXP, un protocole qui permet la remontée d'alarmes en temps-réel dans les RCsF. RTXP se base sur un système de *backoff* et de transmission de *jamming codes* qui permet des accès déterministes au médium et de choisir un seul relayeur lors du routage opportuniste. À notre connaissance, c'est la seule solution qui garantit des contraintes temporelles au niveau MAC et routage, qui est localisée, qui s'adapte au trafic, qui est fiable et efficace en énergie. Nous présentons les bornes théoriques du protocole en termes de délai et de capacité. Par simulation, nous comparons RTXP et PEDAMACS, une solution centralisée. Nous montrons que RTXP est plus adapté à la remontée d'alarmes car il permet une consommation en énergie moindre pour des réseaux de degré moyen inférieur à 50. Les communications radio étant non-fiables, il n'est pas possible de donner des garanties strictes sur les délais de bout en bout. On observe cependant que RTXP offre un meilleur taux de livraison que PEDAMACS. De plus, en comparant favorablement RTXP à une solution non temps-réel, nous montrons l'utilité de l'approche temps-réel même avec de mauvaises conditions radios.

Dans ce chapitre nous déduisons les bornes théoriques de délai et capacité du protocole à partir d'affirmations faites lors de la description du protocole. A partir

de ces affirmations nous construisons des preuves simples des propriétés de RTXP. Cependant, pour pouvoir avoir un niveau de confiance suffisant dans le protocole, il faut le vérifier formellement. En effet, il faut pouvoir être sur qu'une description précise du protocole sous forme de modèle mathématique respecte un ensemble de propriétés voulues. La vérification formelle de RTXP est effectuée dans la deuxième partie de ce document.

Dans ce chapitre nous considérons qu'une seule alarme par événement est envoyée au puits. En réalité, si un événement couvre plusieurs capteurs, cela peut déclencher une tempête d'alarmes qui induit une forte consommation d'énergie dans le réseau et allonge les délais (cela peut même causer des dépassements de la capacité du protocole). Dans le chapitre suivant nous présentons un mécanisme d'agrégation des alarmes qui permet d'atténuer ce problème.

Chapitre 6

Agrégation temps-réel des alarmes : éviter la redondance *

Les applications de détection d'événement visent à apporter l'information au puits qu'un événement d'intérêt a eu lieu dans le réseau (départ d'incendie, glissement de terrain, intrusion, etc). Nous présentons dans le chapitre précédent une solution protocolaire qui permet de réaliser cela en temps borné. Cependant, une des hypothèses pour que cette solution fonctionne est qu'il n'y ait qu'une seule, ou peu d'alarmes qui remontent vers le puits pour un événement donné. En réalité, certains événements, dont l'envergure couvre de multiples capteurs, vont déclencher un nombre très important d'alarmes qui vont être envoyées vers le puits dans un intervalle de temps court. Ce cas est illustré par la Figure 6.1 où tous les nœuds à l'intérieur de la zone où se produit l'événement vont remonter une alarme. Ce phénomène cause une forte charge de trafic dans le réseau, cela induit des collisions et une grande consommation d'énergie. Pour palier à ce problème, nous devons d'agréger les alarmes de chaque nœud se trouvant dans la zone couverte par l'événement, pour réduire le nombre d'alarmes finalement routées vers le puits. Le terme agrégation couvre une grande diversité de mécanismes dans les RCsF qui ont tous pour but de réduire la quantité de données transmises dans le réseau [Krishnamachari et al., 2002]. On peut classer ces mécanismes dans deux catégories [Lu et al., 2010] : l'agrégation spatiale qui consiste à compresser/fusionner des données de plusieurs nœuds et l'agrégation temporelle qui consiste à compresser/fusionner les données au sein d'un nœud. Dans ce chapitre, nous définissons l'agrégation comme un processus qui vise à obtenir un consensus entre les nœuds qui ont détecté un événement pour savoir quels sont ceux qui transmettent effectivement le paquet : on se rapproche donc de la définition de l'agrégation spatiale. Cette agrégation doit se faire en temps borné pour assurer que le délai de détection de l'événement au niveau du puits reste borné.

Dans ce chapitre, nous proposons R^2A (Reliable Real-Time Aggregation), un protocole d'agrégation temps-réel basé sur les coordonnées virtuelles présentées dans le Chapitre 4 et utilisées dans RTXP. Ce protocole permet de réduire le nombre

∗. Ce travail est issu des travaux de Master 2 Recherche de Xuan Linh Nguyen que j'ai co-encadrés avec Isabelle Augé-Blum.

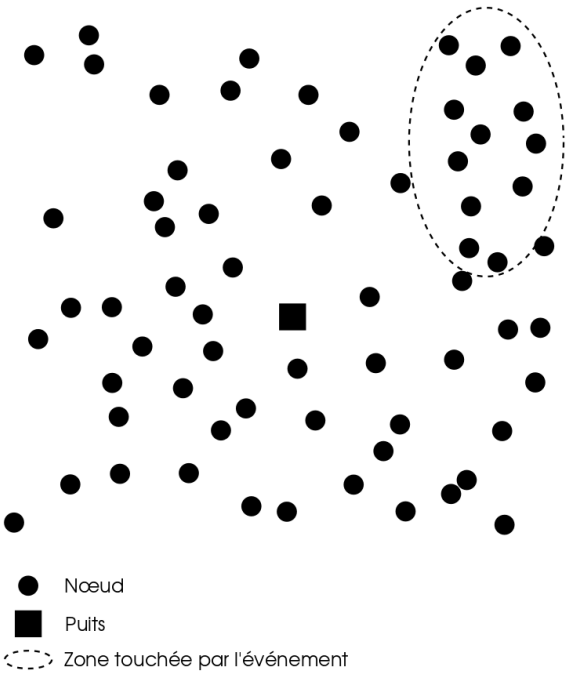


FIGURE 6.1 – Tous les nœuds inclus dans la zone d'événement vont envoyer une alarme

d'alarmes effectivement envoyées au puits en échangeant seulement des messages non explicites (*jamming codes*) ce qui le rend robuste aux mauvaises conditions radio.

6.1 Etat de l'art sur l'agrégation dans les RCsF

Le principe général de l'agrégation dans les RCsF consiste à diminuer le nombre de transmissions en en regroupant certaines spatialement et/ou temporellement [Krishnamachari et al., 2002] [Lu et al., 2010] :

- agrégation temporelle : processus qui permet de regrouper les informations captées ou reçues par un nœud donné pendant une durée déterminée.
- agrégation spatiale : processus qui permet de regrouper les informations captées par les nœuds dans une zone donnée à un instant donné.

Les protocoles d'agrégations existants présentent souvent les deux aspects, spatial et temporel. On peut donc les classer selon l'étendue du processus d'agrégation [Fan et al., 2006]. On distingue trois cas : les protocoles qui utilisent une structure globale d'agrégation qui correspond à une organisation logique des nœuds (par exemple en arbres ou en *clusters*), ceux qui n'utilisent pas de structure et ceux qui utilisent une structure locale d'agrégation autour de la zone de détection de l'événement.

Dans ce chapitre, nous utilisons cette dernière classification pour présenter les différents protocoles d'agrégation de l'état de l'art. Nous évoquerons cependant les autres critères de classification (temporelle et spatiale) dans les descriptions des protocoles. De plus, nous nous intéressons plus particulièrement au type d'agrégation qui

permet d'éviter la redondance d'information de détection d'un événement. C'est donc un type d'agrégation local à l'événement et qui n'implique pas de traitement sur les données, nous détaillerons donc ces types de solutions au cours de l'état de l'art.

6.1.1 Protocole d'agrégation avec structure globale

Pour les protocoles organisés en structures globales, les nœuds du réseau s'organisent en arbre ou bien en *clusters* comme dans le cas des protocoles de routage hiérarchiques décrits dans le Chapitre 2 dans la section 2.1.2. Les protocoles décrits dans cette section (2.1.2) sont, en effet, non seulement des protocoles de routage car ils permettent de trouver des routes entre les nœuds sources et le puits, mais peuvent aussi avoir la fonction d'agrégation car les données de plusieurs nœuds d'un même niveau hiérarchique peuvent être traitées et/ou regroupées avant d'être transmises à un niveau plus proche du puits. Prenons, par exemple, LEACH [Heinzelman et al., 2000] qui est un protocole qui organise le réseau en *clusters*. On constate que le *cluster head* récupère les données de tous les membres du *cluster* avant de les transmettre au puits, de manière regroupée (il peut éventuellement éliminer les redondances de données suivant l'application).

Cependant, ce type de structure globale n'est pas construite en fonction des événements qui apparaissent dans le réseau. Un événement peut donc couvrir de nombreux *clusters* ou bien des branches éloignées d'un arbre, ce qui fait que des paquets contenant exactement la même information vont être transmis un nombre important de fois avant d'être effectivement agrégés.

Il existe de nombreux autres protocoles hiérarchiques [Al-Karaki et al., 2009] [Sardouk et al., 2010] [IETF-ROLL, 2011], nous nous focalisons cependant, d'avantages sur les solutions locales à un événement qui répondent mieux à notre problématique.

6.1.2 Protocole d'agrégation avec structure locale

Les protocoles d'agrégation avec structures locales forment une structure provisoire à la détection d'un événement. Cette structure est centrée sur l'événement et est locale à l'événement, contrairement au cas des structures globales qui sont centrées sur le puits et recouvrent tout le réseau.

EDMDP [Xu et al., 2009] est un protocole d'agrégation qui construit des *clusters* autour des événements détectés dans le réseau. La taille d'un *cluster* et sa durée de vie sont fonctions de la gravité de l'événement. Cette gravité est déterminée en fonction de l'écart entre la valeur du paramètre mesuré par le capteur et une valeur de référence. Le *cluster* est construit grâce à un message d'initialisation envoyé en *broadcast*. Les membres du *cluster* envoient ensuite leurs données au *cluster-head* qui les agrège et envoie un seul message vers le puits. Ce type de protocole est adapté à la réduction de redondance lors de la détection d'événements. Cependant certains points ne sont pas abordés par les auteurs. Par exemple, l'élection du *cluster-head* n'est pas détaillée. Les accès au canal ne sont pas non plus détaillés. Il n'est donc pas possible de garantir que les délais de bout en bout sont bornés avec cette solution,

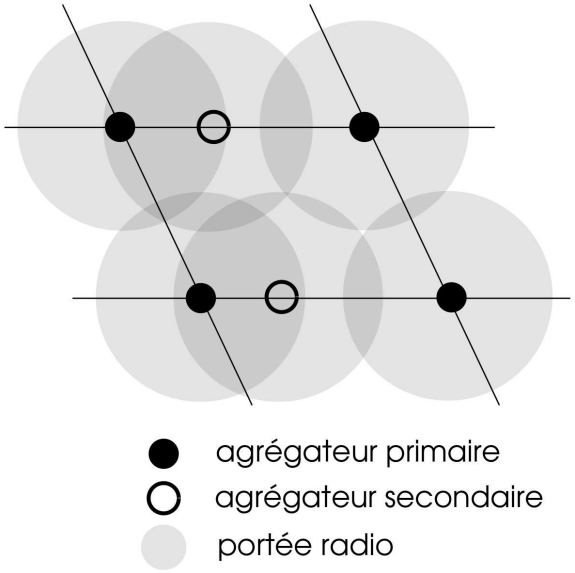


FIGURE 6.2 – Illustration du découpage en parallélogrammes du protocole SFEB, les agrégateurs primaires et secondaires couvrent toute la surface.

car les auteurs ne garantissent pas que la formation des *clusters*, les accès au médium pour remonter les mesures au *cluster-head* et l'envoi des données du *cluster-head* au puits se font en temps borné.

Dans SFEB [Chao and Hsiao, 2013], les nœuds connaissent leurs coordonnées géographiques. Comme illustré par la Figure 6.2, le réseau est découpé en parallélogrammes de côtés $(1+\sqrt{3})R$ et $R\sqrt{3}$ avec R le rayon de communication d'un nœud. Quand un événement est détecté, les nœuds à moins de $\frac{1}{2}R$ de l'angle d'un parallélogramme peuvent concourir pour devenir agrégateur primaire (équivalent d'un *cluster-head*). Un agrégateur secondaire est sélectionné de la même manière, mais pour des positions éloignées de l'angle d'une distance R vers la droite (ils permettent de couvrir les zones non couvertes par les agrégateurs primaires comme illustré par la Figure 6.2). Ces deux agrégateurs récupèrent ensuite le trafic de leurs voisins. Pour cela, les agrégateurs connaissent le nombre de voisins qu'ils possèdent, ils découpent leur voisinage en zones géographiques. Les nœuds d'une zone entrent en contention pour accéder au canal de manière aléatoire. La récupération des données des zones est ordonnancée de façon à minimiser les interférences inter-zones. Une fois les données agrégées au niveau des agrégateurs primaires et secondaires, elles sont remontées vers le puits en commençant par les agrégateurs les plus éloignés du puits. Cette solution ne permet pas une agrégation et une remontée des données vers le puits en temps borné, car les accès au canal se font de manière aléatoire. De plus l'élection des agrégateurs demande un nombre important de transmissions, induisant un coût énergétique non négligeable.

Les protocoles présentés dans cette section permettent d'agréger les paquets issus d'un événement de manière locale. Cependant, la création d'une structure logique

d'agrégation à chaque événement rend ce type de solution gourmande en énergie et augmente les délais de bout en bout sans pouvoir les borner.

6.1.3 Protocole d'agrégation sans structure

Dans le cas des protocoles dits sans structure, l'agrégation ne se fait pas en fonction d'une organisation logique du réseau en structure (*clusters*, arbre), mais en fonction de coordonnées des nœuds. Ces coordonnées pouvant être réelles ou virtuelles comme évoqué dans le Chapitre 2. On peut avancer que les coordonnées virtuelles sont un type de structure car elles sont construites en fonction des liens entre les nœuds. Cependant, dans le cas d'une structure, un nœud a une connaissance explicite de ses voisins dans la structure, la coordonnée virtuelle ne donne qu'une information sur la position globale du nœud dans le réseau sans lui assigner un *cluster-head*, des fils ou des parents. Ce type d'approche est donc plus robuste [Ye et al., 2005a] : on peut perdre quelques liens sans avoir à recalculer la coordonnée (cas du gradient), alors que si on perd le lien avec son père dans un arbre, il faut réorganiser tout ou partie de l'arbre.

DA-MAC [Wu et al., 2012] est un protocole *cross-layer* (MAC, routage et agrégation) sans structure pour RCsF. Comme dans le cas de GRAB [Ye et al., 2005a] ou bien RTXP, chaque nœud connaît le nombre de sauts qui le sépare du puits. Cette information peut être vue comme une coordonnée virtuelle à une dimension. Au niveau MAC, DA-MAC reprend le mécanisme de préambule découpé en paquets de X-MAC. Cependant, dans le cas de DA-MAC il n'y a pas d'adresse de destination dans les paquets de préambule, mais seulement le nombre de sauts qui sépare l'émetteur du puits. Un nœud plus près du puits que l'émetteur reste éveillé durant le préambule jusqu'à recevoir la donnée. Tandis qu'un nœud plus éloigné du puits peut répondre aux paquets de préambule pour stopper l'envoi et envoyer son paquet de données avant. De cette manière, les données les plus éloignées du puits sont envoyées d'abord et cela permet de les agréger avec celles provenant de nœuds plus proches du puits. Cette solution permet de réaliser à la fois l'agrégation et le routage. Cependant, les accès au médium se font en CSMA, cela ne permet pas de garantir des délais de bout en bout bornés.

Dans [Kai-Wei Fan et al., 2007], les auteurs proposent un protocole d'agrégation sans structure basé sur la position géographique des nœuds. Pour réaliser l'agrégation spatiale des données, le protocole DAA [Kai-Wei Fan et al., 2007] est utilisé : un nœud qui a un paquet à transmettre envoie un RTS [IEEE, 2003], les voisins les plus proches du puits et ayant déjà un paquet similaire (issu de la détection du même événement) sont prioritaires pour répondre (avec un CTS). Pour assurer la convergence temporelle, les nœuds attendent un temps aléatoire avant de retransmettre un paquet. Ce mécanisme ne permet pas une remontée en temps-réel des alarmes vers le puits, car il utilise un routage géographique et des accès aléatoires au médium. Comme nous l'avons évoqué dans le Chapitre 2, ce type de solution ne permet pas de garantir des délais bornés. Les auteurs de [Yousefi et al., 2012] ajoutent la prise en compte des contraintes temporelles dans le choix des relayeurs et du temps d'attente dans un nœud. Ils utilisent pour cela la métrique de vélocité de SPEED

[Stankovic and Abdelzaher, 2003]. Comme nous l'avons vu dans le Chapitre 2, même si cela permet d'améliorer le taux d'échéances respectées, cela ne permet pas de borner les délais de bout en bout.

Les protocoles d'agrégation sans structure ont l'avantage d'être plus robustes que ceux avec structure, car tous les voisins d'un nœud sont des récepteurs (agrégateurs) potentiels. Ce type de protocole permet aussi de réaliser une agrégation locale à un événement. Cependant, les protocoles proposés dans la littérature allient agrégation à l'échelle de l'événement et routage, mais sans donner de garanties sur les délais de bout en bout. Dans les sections suivantes, nous proposons une solution d'agrégation basée sur les coordonnées présentées dans le Chapitre 4 qui permet de réaliser une agrégation locale à l'événement mais sans réaliser le routage, qui est laissé à RTXP (Chapitre 5).

6.2 Proposition d'un protocole d'agrégation temps-réel : R^{2}A

Dans cette section, nous présentons R^{2}A, un protocole d'agrégation qui permet de sélectionner, parmi les nœuds qui détectent un événement, celui ou ceux (peu dans l'idéal) qui envoient effectivement un paquet au puits pour signaler l'événement. Cela permet d'éviter l'envoi d'informations redondantes, qui charge le réseau, augmente le taux de collisions et induit une surconsommation d'énergie. Ce protocole réalise l'agrégation des données durant les périodes de sommeil de RTXP, cela permet de ne pas ajouter de délais supplémentaires au délai de bout en bout.

6.2.1 Hypothèses

Nous reprenons les hypothèses de RTXP :

- Les nœuds ont des capacités limitées (mémoire, énergie).
- Les radios des nœuds sont *half-duplex*.
- On considère un modèle d'interférences à deux sauts : les nœuds communiquent avec leurs voisins et interfèrent avec leurs 2-voisins.
- Les nœuds sont munis d'une coordonnée correspondant à celle présentée dans le Chapitre 4.

6.2.2 Modélisation des événements

Pour pouvoir concevoir et évaluer le protocole d'agrégation des paquets concernant un événement, il faut pouvoir modéliser les événements. Nous utilisons le modèle d'événements présent dans le simulateur WSNet [WSNet, 2009] : nous considérons qu'un événement couvre une zone circulaire dés son apparition. Ensuite, l'événement se propage dans toutes les directions avec la même vitesse v. Il conserve donc une forme de disque. La surface couverte par l'événement à un instant t est donnée par :

$$S = \pi[r_0 + v(t - t_0)]^2 \tag{6.1}$$

avec t_0 l'instant d'apparition de l'événement et r_0 le rayon initial. On fait aussi l'hypothèse que la propagation des messages du protocole d'agrégation est plus rapide que celle de l'événement, cette hypothèse est réaliste dans la mesure où les phénomènes observés ne se déplacent pas plus vite que la valeur de la portée radio divisée par le temps de transmission d'un paquet (par exemple, la vitesse de propagation d'un feu de forêt varie de 4 à 6 km/h [Wikipedia, 2013c] et pour un débit de 125kbps, un paquet de 200 bits et une portée de 100m, on obtient une vitesse de propagation de 2.3×10^5km/h donc largement supérieure à celle du phénomène observé).

Ce modèle très simple, nous donne une approximation de la propagation d'un phénomène physique (un feu par exemple) et nous permet de concevoir et tester notre protocole. Dans des travaux futurs, nous prévoyons d'utiliser des données réelles d'événements (feux de forêt, nuages de pollution, etc) pour évaluer notre proposition dans des cas plus réalistes.

6.2.3 Vue d'ensemble de la proposition

R^{2}A est basé sur les coordonnées présentées dans le Chapitre 4. Comme illustré sur la Figure 6.3, le protocole d'agrégation s'exécute seulement durant les périodes de sommeil de RTXP. Sur détection d'un événement les nœuds endormis se réveillent et exécutent R^{2}A (si les nœuds sont en train d'exécuter une période active de RTXP, ils attendent la période de sommeil pour agréger - ce délai est pris en compte dans le délai pire cas de RTXP, cf. Equation 5.6 p.93). R^{2}A réalise deux opérations : il synchronise les nœuds qui détectent un même événement et leurs permet de se départager pour sélectionner lequel d'entre eux envoie un paquet. Pour accomplir ces deux tâches, le protocole utilise seulement des messages non explicites, des *jamming codes*. On note que la période active de RTXP ne peut pas être retardée par l'agrégation pour conserver la propriété temps-réel de RTXP, il est donc possible que l'agrégation n'ait pas le temps de finir. Plusieurs nœuds envoient alors une alarme vers le puits, l'impact de ce comportement est étudié dans la section 6.3.

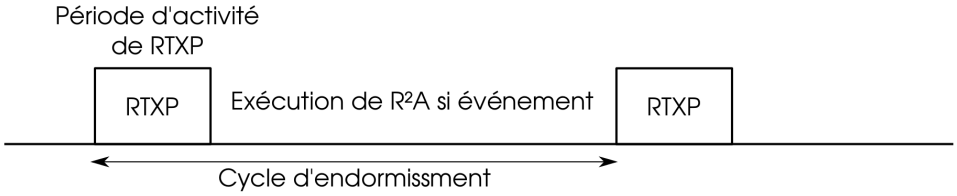


FIGURE 6.3 – R^{2}A s'exécute seulement durant les périodes de sommeil de RTXP, sur détection d'un événement

Synchronisation

Lorsqu'un événement survient dans le réseau, il peut être détecté par plusieurs nœuds au même instant. Les nœuds qui détectent l'événement en premier activent leur radio. Comme illustré par la Figure 6.4, ils utilisent une temporisation durant laquelle ils échantillonnent le canal pour vérifier qu'ils sont effectivement les premiers :

la temporisation est nommée *sync timer*. Si les nœuds ne détectent aucune activité radio avant l'expiration de la temporisation, ils en déduisent qu'ils sont les premiers à détecter l'événement et émettent périodiquement un *jamming code* long que nous nommons *jamming code* P (dans la suite de ce Chapitre, *jamming code* P désigne un *jamming code* long et *jamming code* désigne un *jamming code* court). La période d'émission des *jamming code* P est égale à la durée de la temporisation. Ce *jamming code* P a pour but de synchroniser les nœuds qui se réveillent plus tard, comme illustré sur la Figure 6.4 : A et B détectent l'événement dès son apparition, à l'expiration de leur temporisation *sync timer* ils émettent un *jamming code* P. C détecte l'événement plus tard, il arme sa temporisation *sync timer* mais détecte un *jamming code* P avant la fin de la temporisation, cela signifie qu'il n'est pas le premier à détecter l'événement ; il va donc se synchroniser avec A et B grâce au *jamming code* P qu'il détecte (il en émettra ensuite périodiquement pour synchroniser ses éventuels voisins).

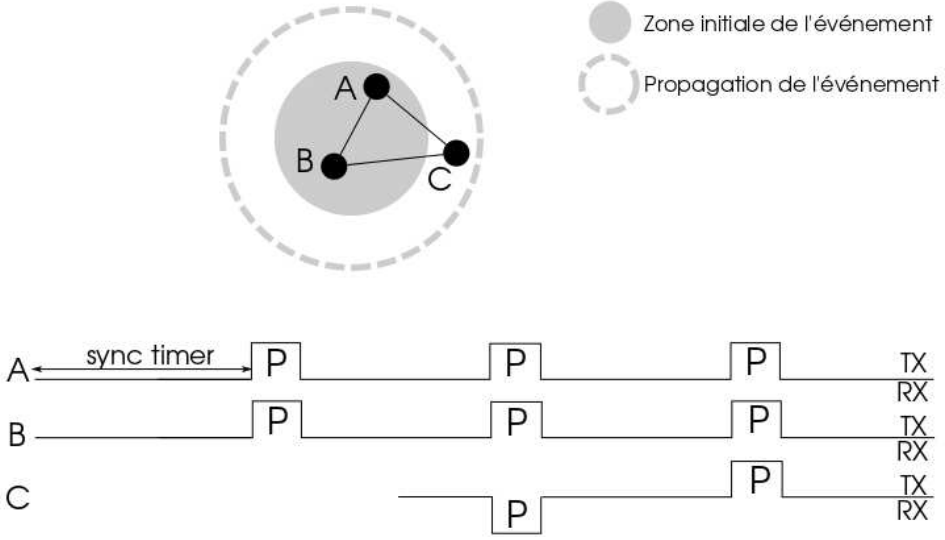


FIGURE 6.4 – Exemple de synchronisation : A et B détectent le début de l'événement, C est ensuite atteint et se synchronise avec A et B

Tous les nœuds émettent ensuite périodiquement un *jamming code* P pour synchroniser les nœuds qui se réveillent plus tard, au fur et à mesure de la propagation de l'événement. On note que le *jamming code* P peut être vu comme un *beacon* de 802.15.4 [IEEE, 2011b] mais ne contenant pas d'information explicite (cela autorise plusieurs nœuds interférants à l'émettre simultanément).

Election de l'expéditeur du paquet

Entre les *jamming codes* P, les nœuds éveillés entrent en contention pour élire lequel d'entre eux sera l'émetteur du paquet. Cette phase de contention fonctionne de façon similaire à celles de RTXP (voir Chapitre 5 section 5.5). Chaque nœud calcule son temps de *backoff* en fonction de sa coordonnée. Si le *backoff* d'un nœud expire et qu'il n'a pas détecté de *jamming code*, il émet un *jamming code*. S'il détecte

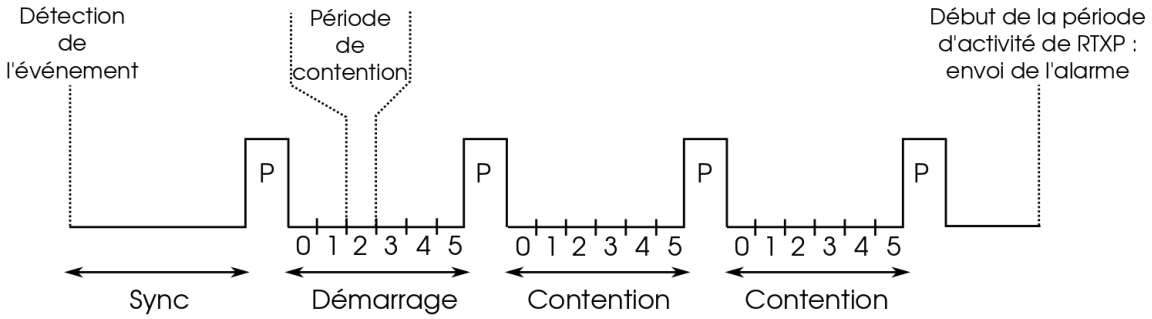


FIGURE 6.5 – Comportement d'un nœud exécutant R^{2}A entre l'instant de détection de l'événement et l'instant d'envoi de l'alarme (qui correspond au début d'une période d'activité de RTXP).

un *jamming code*, cela signifie qu'un nœud est plus prioritaire que lui pour émettre le paquet d'alarme. Il répète le *jamming code* du gagnant dès la réception des premiers bits pour informer des nœuds plus éloignés qu'il y a un gagnant. On remarque que l'utilisation de messages non-explicites permettent d'atteindre une fiabilité élevée [Boano et al., 2012] (les paquets n'ont pas besoin d'être décodés), cependant il n'est pas possible avec cette solution de discriminer deux événements de natures différentes qui surviendraient dans la même zone géographique (R^{2}A traite ce cas comme un seul événement).

6.2.4 R^{2}A en détails

Le fonctionnement détaillé de R^{2}A est décrit par la Figure 6.5. D'abord, le nœud qui détecte un événement se synchronise avec les nœuds éventuellement déjà éveillés durant la phase *Sync* décrite dans la section précédente. On distinguera par la suite le comportement des nœuds qui détectent l'événement en premier (comme A et B sur la Figure 6.4) des autres. Après s'être synchronisé, un nœud exécute deux autres phases, la phase de démarrage et la phase de contention. Ces deux phases permettent de déterminer le nœud émetteur du paquet alarme. Ce sont des phases de contention, elles se situent entre les émissions périodiques des *jamming codes* P et sont divisés en périodes de contention. Les nœuds de gradients égaux entrent en contention dans la même période. Comme nous supposons un modèle d'interférence à 2 sauts, les nœuds à 5 sauts d'écart peuvent utiliser la même période (pour ne pas interférer avec les nœuds de gradients inférieurs et supérieurs). Il est donc possible de réutiliser une période tous les 5 niveaux, dans R^{2}A nous réutilisons une période tous les 6 niveaux pour garder une marge de sécurité qui limite les interférences. La durée entre 2 *jamming codes* P est donc divisée en 6 périodes de contention égales comme représenté sur la Figure 6.5. Un nœud étant à h sauts du puits utilise la période $h \mod 6$. En scrutant l'activité dans les autres périodes de contention, un nœud peut apprendre si des nœuds plus proches ou plus éloignés que lui du puits ont détecté l'événement.

Phase de démarrage

Après la synchronisation (émission ou réception de son premier *jamming code* P), le nœud passe en phase de démarrage. Cette phase permet au nœud (de gradient h) de déterminer s'il est :

- Gagnant : nœud qui ne détecte aucun nœud plus prioritaire (pas de *jamming code* dans les périodes de contention correspondants à $h-2$, $h-1$ et gagne la contention dans la période correspondant à h). Les nœuds gagnants à la fin de l'agrégation envoient un message d'alarme au puits.
- Quasi-gagnant : nœud qui détecte un *jamming code* dans la période correspondant au gradient $h-1$ mais pas dans celui correspondant à $h-2$. Il n'enverra pas de message d'alarme mais il répète les *jamming codes* du gagnant pour propager l'information qu'il y a un gagnant et ainsi limiter le nombre de gagnants (comme on le verra lors de la description de la Figure 6.6 plus loin dans cette section).
- Perdant : un nœud qui détecte un nœud plus prioritaire dans la période correspondant au gradient $h-2$. Il n'enverra pas de message d'alarme et ne participe plus au mécanisme d'élection.

Pour connaître leurs états, les nœuds ont un comportement qui dépend de s'ils ont détecté l'événement en premier ou non (s'ils sont dans le cas de A et B de la Figure 6.4 ou bien de C) :

Nœuds détectant l'événement en premier. Dans ce cas, les nœuds vont entrer en contention dans la phase de démarrage. Dans sa période de contention, c'est-à-dire celui correspondant à $h \mod 6$, un nœud lance son *backoff* pour un temps qui dépend de sa coordonnée. Si le *backoff* expire, il émet un *jamming code*, s'il détecte un *jamming code* avant l'expiration de son *backoff*, il le répète aussitôt (dés la réception des premiers bits) pour propager aux nœuds moins prioritaires le fait qu'il y a déjà un nœud gagnant pour cet événement.

Dans les autres périodes, un nœud scrute l'activité. Dans les périodes correspondants aux nœuds plus proches du puits ($[(h-1) \mod 6]$ et $[(h-2) \mod 6]$), un nœud qui détecte un *jamming code* le répète pour propager l'information. Pour les périodes des nœuds les plus éloignés du puits ($[(h+1) \mod 6]$ et $[(h+2) \mod 6]$) le nœud ne fait rien. La répétition pour la propagation de l'information est utile pour limiter le nombre de nœuds gagnants. Par exemple, avec la topologie représentée sur la Figure 6.6, sans répétition D et G qui ne détectent pas de *jamming code* de A (ils sont à plus de deux sauts de A) vont croire qu'ils sont des nœuds gagnants, alors que seul A devrait l'être (il est le plus proche du puits). La répétition du *jamming code* par les autres nœuds prévient ce cas de figure.

A la fin de cette phase, les nœuds peuvent déterminer s'ils sont gagnants quasi-gagnants ou perdants.

Nœuds détectant l'événement après propagation. Dans ce cas, le nœud se réveille et détecte un *jamming code* P. Le nœud écoute ensuite ce qui se passe dans toutes les périodes de contention pour déterminer si des nœuds ont déjà détectés l'événement et quelles sont leurs positions par rapport au puits : si un nœud détecte des *jamming codes* P mais pas de *jamming code* courts dans les périodes de contention,

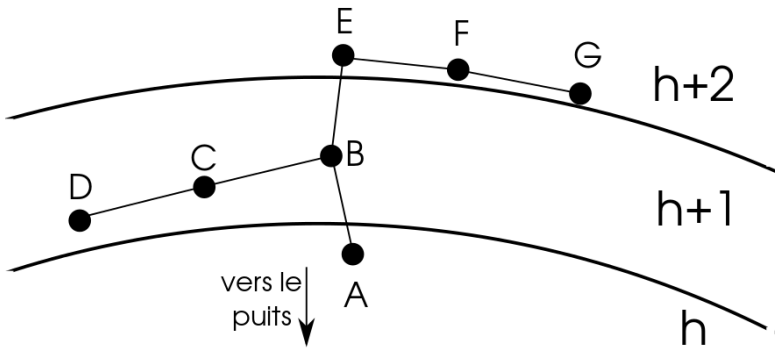


FIGURE 6.6 – Dans cette configuration, si G est plus prioritaire que E et F ; et D est plus prioritaire que C et B ; si B et E ne répètent pas le *jamming code* de A, alors G et D vont se considérer comme gagnants.

il se considère comme perdant (les nœuds, dans ce cas, n'émettent pas de *jamming code* durant la phase de démarrage). S'ils détectent des *jamming codes* court dans les périodes de contention, ils définissent leurs états (gagnant, quasi-gagnant, perdant) en fonction des critères précédemment énoncés.

Phase de contention

La fin de la phase de démarrage est marquée par un *jamming code* P. Durant la phase de contention, les nœuds agissent en fonction de l'état dans lequel ils se trouvent à la fin de la phase de démarrage :

- les nœuds gagnants émettent un *jamming code* dans leur période de contention ;
- les nœuds quasi-gagnants répètent les *jamming code* des nœuds de gradient $h-1$;
- les nœuds perdants se rendorment pour le temps de cette phase (mais continuent à émettre des *jamming P*).

A la fin de cette phase, les nœuds mettent de nouveau à jour leur état en fonction des événements de la période de contention. Cette période se répète jusqu'à l'envoi du paquet.

Exemple

La Figure 6.7 illustre un exemple d'exécution du protocole R^2A. Lors de son apparition, l'événement couvre une zone correspondant au disque gris. Il s'étend ensuite au cercle gris en pointillés.

- Les nœuds B, C, D, E, F et G détectent l'événement en premier et simultanément lors de son apparition. Ces nœuds attendent pendant un *sync timer* pour vérifier qu'ils sont effectivement les premiers à détecter l'événement. A l'expiration de cette temporisation, ils envoient un *jamming code* P.
- Ces nœuds exécutent ensuite leur phase de démarrage. Etant donné qu'ils détectent l'événement en premier, ils entrent en contention dans leur période de

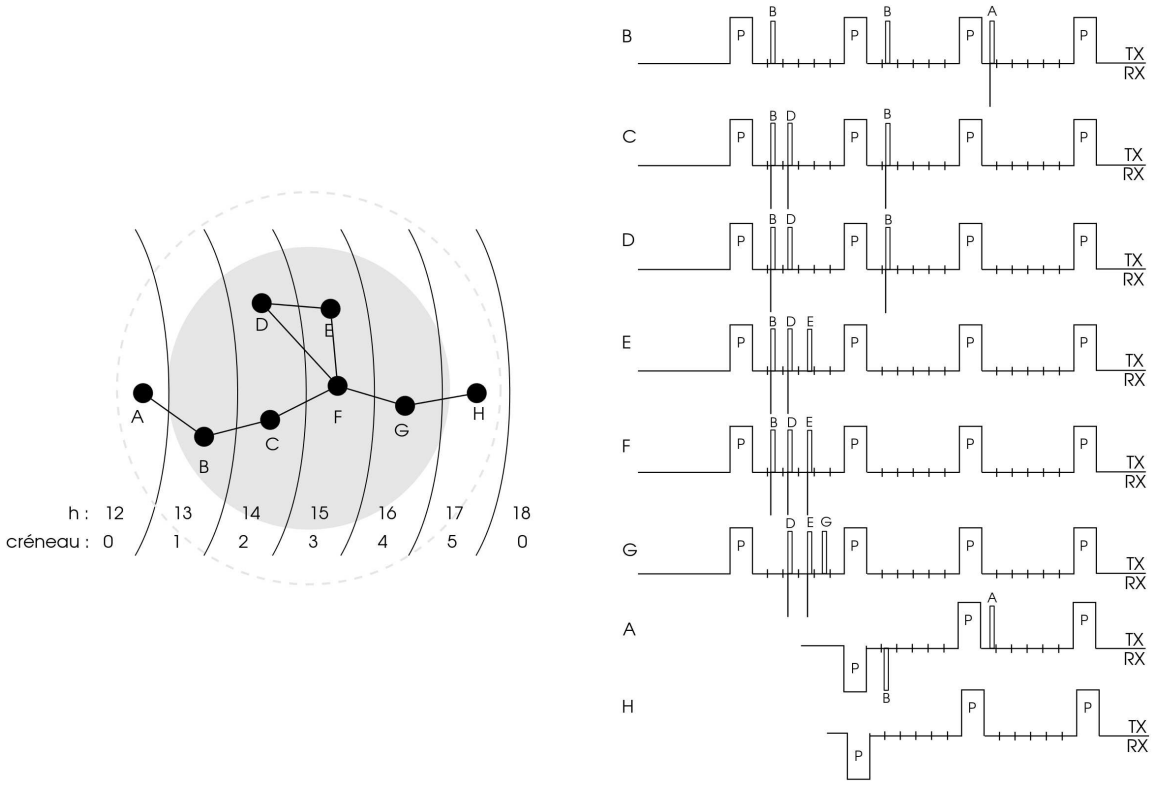


FIGURE 6.7 – Exemple d'exécution de R2A

contention ($h \mod 6$). Dans une période donnée, le nœud dont le *backoff* expire en premier émet un *jamming code* qui est répété par les nœuds à $[(h+1) \mod 6]$ et $[(h+2) \mod 6]$. C'est le cas des nœuds B, D, E et G dont les *jamming codes* sont répétés par les nœuds C, D, E, F pour B, par les nœuds C, E, F, G pour D, par les nœuds F, G pour E et n'est pas répété pour G. A la fin de cette phase, le nœud B est gagnant, les nœuds C et D sont quasi-gagnants et les autres sont perdants.

- Dans la phase suivante, c'est-à-dire la phase de contention, B qui est gagnant, envoie un *jamming code* dans sa période de contention. Il est répété par les nœuds quasi-gagnants C et D.
- Pendant la phase de démarrage des nœuds B, C, D, E, F et G l'événement se propage et est détecté par les nœuds A et H. Ces nœuds, à la détection de l'événement, échantillonnent le canal pour savoir s'ils sont les premiers. Ils détectent un *jamming code* P, ce qui les informe qu'il ne sont pas les premiers. Après la réception du *jamming code* P, A et H entrent en phase de démarrage.
- Comme ils n'ont pas détecté l'événement en premier, ils n'entrent pas en contention pendant leur phase de démarrage, mais écoutent le canal. H ne détecte de *jamming codes* dans aucune période de contention. Il en déduit qu'il est perdant. A détecte un *jamming code* dans la période qui correspond aux nœuds un saut plus loin que lui du puits (celui de B). Il est donc gagnant à la fin de sa phase de démarrage.

- Dans la phase suivante, la phase de contention, A envoie son *jamming code* dans sa période, il est répété par B qui passe alors en état quasi-gagnant. Les autres nœuds sont perdants et ne participent donc pas à cette phase.

Dans la section suivante, nous évaluons la capacité de R^{2}A à réduire le nombre d'alarmes envoyées au puits et le gain en énergie que cela permet.

6.3 Evaluation des performances de R^{2}A

Les performances de R^{2}A sont évaluées par simulation. Comme dans les Chapitre 4 et 5, nous utilisons le simulateur à événements discrets WSNet [WSNet, 2009]. Les paramètres de R^{2}A observés sont :

- Le nombre d'alarmes envoyées par rapport au nombre de nœuds qui détectent l'événement.
- La position des nœuds émetteurs par rapport aux nœuds détectant l'événement les plus proches du puits.

Ensuite, les performances de RTXP seul et RTXP avec R^{2}A sont comparées. Les paramètres observés sont le délai de bout en bout et l'énergie dépensée.

6.3.1 Paramètres et scénarios de simulation

Paramètre	Valeur
Nombre de nœuds	500 - 600
Débit	500kbps
Portée radio	10 unités
Surface de déploiement	100×100 unités
Taille d'un paquet	100 octets
Durée d'un *jamming code*	100 μs
Durée d'un *jamming code* P	200 μs
Rapport du cycle d'endormissement de RTXP	1%
Vitesse de propagation de l'événement	$k \times 0.3$ unités/s, $k \in [0, 5]$
Taille initiale de l'événement	$k \times 0.3$ unités, $k \in [0, 5]$

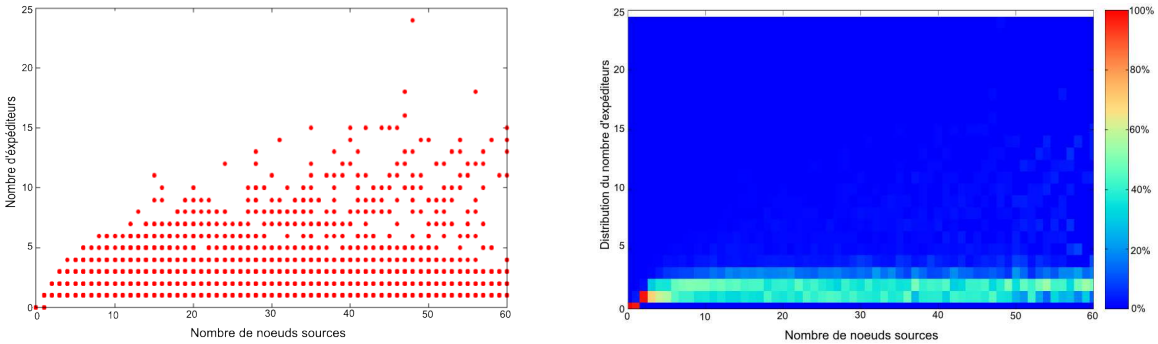
Table 6.1 – Paramètres de simulation

Pour la campagne de simulation, nous avons généré 40 topologies aléatoires pour lesquelles les nœuds sont déployés dans une aire de 100×100 unités avec le puits au centre. Pour chaque topologie, 10 simulations sont exécutées. Dans chaque simulation, 360 événements sont générés aléatoirement. Ils ont des tailles initiales, des vitesses de propagation et des positions d'apparition différentes (c.f. Tableau 6.1). Les instants de début des événements sont choisis de manière aléatoire. Durant les simulations, un événement disparaît à la réception du premier paquet d'alarme le concernant par le puits car on évalue seulement la détection de l'événement et non son observation périodique. En revanche, les paquets apparus à cause de cet événement ne disparaissent

pas et doivent être traités par le réseau. Les capteurs ont une fréquence d'échantillonnage de 1Hz. Comme pour les simulations du Chapitre précédent, nous utilisons les modèles de propagation de pertes en espace libre et de *log-normal shadowing*.

6.3.2 Performances de R^{2}A

Nous nous intéressons d'abord au pouvoir d'agrégation de R^{2}A, c'est-à-dire sa capacité à avoir un nombre d'émetteurs d'alarmes beaucoup plus petit que le nombre de nœuds qui détectent l'événement.

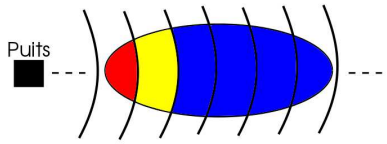


(a) Nombre d'émetteurs (b) Distribution du nombre d'émetteurs

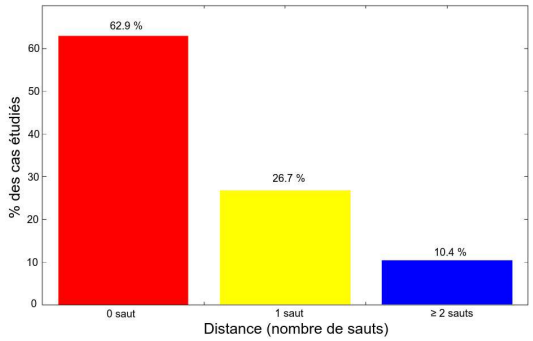
FIGURE 6.8 – Etude du nombre d'émetteurs du paquet par rapport au nombre de sources

La Figure 6.8(a) représente le nombre de nœuds émetteurs (qui envoient une alarme) en fonction du nombre de nœuds sources (qui détectent l'événement). On constate qu'au delà de 20 nœuds sources, le nombre d'expéditeurs n'augmente pas en fonction du nombre de sources. Dans la plupart des cas, le nombre d'expéditeurs maximum est en dessous de 15. La Figure 6.8(b) illustre la distribution du nombre d'émetteurs en fonction du nombre de nœuds sources : pour chaque nombre de nœuds sources, on effectue plusieurs simulations et on observe la distribution des résultats (nombre de nœuds émetteurs). On constate que dans la plupart des cas le nombre d'émetteurs est en dessous de 3 (dans 80% des cas). Les résultats de la Figure 6.8 sont donnés pour un modèle de propagation de pertes en espace libre. On peut noter que les résultats pour le modèle *log-normal shadowing* sont très similaires, car R^{2}A utilise exclusivement des messages non explicites qui sont robustes aux atténuations et aux interférences [Boano et al., 2012].

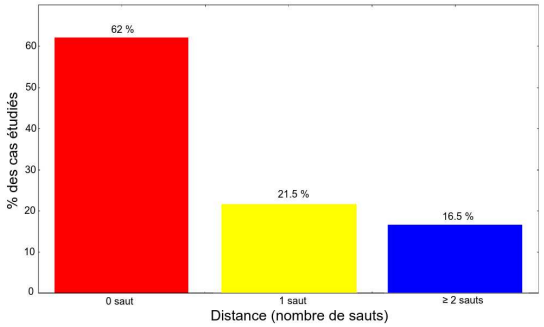
Lorsqu'un événement apparaît et se propage dans le réseau, il peut couvrir des nœuds qui ont des valeurs de gradient h différentes. Pour diminuer le délai de bout en bout et l'énergie consommée, il est préférable que des nœuds parmi les plus proches du puits soient sélectionnés pour effectivement envoyer l'alarme. La Figure 6.9 représente la répartition des émetteurs en fonction de la distance en nombre de sauts au nœud détectant l'événement le plus proche du puits. Sur la Figure 6.9(a), les nœuds détectant l'événement sont classés en trois zones : les nœuds les plus proches du puits, les nœuds éloignés d'un saut des plus proches et les nœuds éloignés au moins de deux



(a) Représentation du découpage de l'événement en fonction du nombre de sauts séparant les nœuds du puits



(b) Cas du modèle de pertes en espace libre



(c) Cas du modèle *log-normal shadowing*

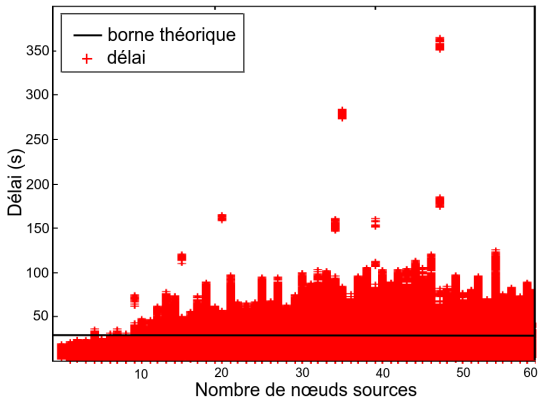
FIGURE 6.9 – Répartition des distances entre le nœud émetteur de l'alarme et le nœud détectant l'événement le plus proche du puits

sauts des plus proches. On constate que, quel que soit le modèle de propagation (Figure 6.9(b) pour le modèle de pertes en espace libre et Figure 6.9(c) pour le modèle *log-normal shadowing*), dans quasiment deux tiers des cas simulés, c'est un nœud de la zone la plus proche du puits qui émet l'alarme. Dans environ un quart des cas, c'est un nœud à un saut du puits qui émet l'alarme. Dans le reste des cas, c'est un nœud à au moins deux sauts (ce sont des cas où R^{2}A n'a pas le temps de terminer l'agrégation avant le début de la période d'activité de RTXP). Le fait que la plupart des alarmes soit émises de nœuds les plus proches possible du puits réduit la consommation d'énergie et le délai de bout en bout (moins de sauts à effectuer).

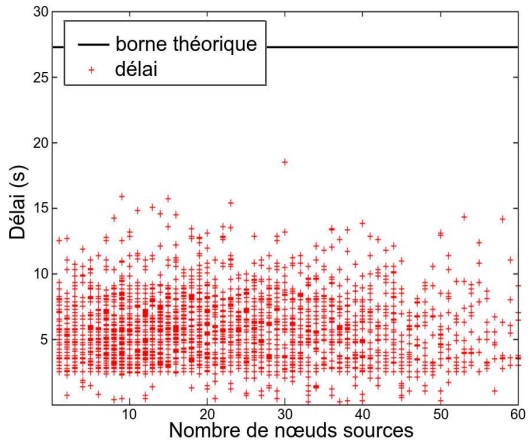
6.3.3 Comparaison entre RTXP seul et RTXP avec R^{2}A

Dans cette section, nous comparons les performances de RTXP seul avec celles de RTXP associé à R^{2}A. Cela permet de vérifier que R^{2}A apporte en effet une amélioration des performances de RTXP : nous évaluons le délai de bout en bout pour observer si l'utilisation de R^{2}A permet de respecter les échéances temporelles et l'énergie consommée pour vérifier que l'utilisation de R^{2}A augmente la durée de vie du réseau en réduisant le nombre d'informations redondantes acheminées.

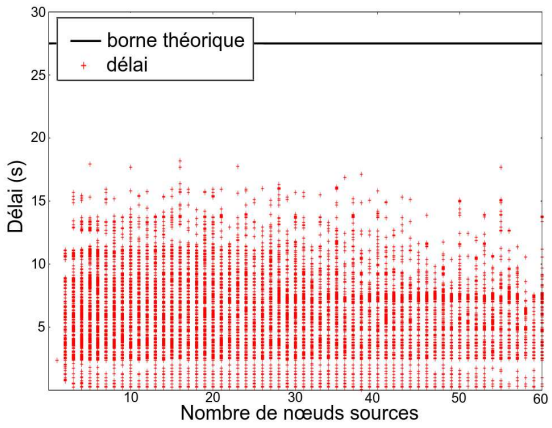
Les Figures 6.10(a), 6.10(b) et 6.10(c) représentent respectivement les délais de RTXP seul, du premier message de RTXP et de RTXP avec R^{2}A. Sur la Figure 6.10(a) on constate qu'un nombre important d'alarmes sont reçues par le puits (une croix correspond au délai d'une alarme). La plupart de ces alarmes arrivent au puits après



(a) Délais pour RTXP seul (tous les messages)



(b) Délais du premier message de chaque événement pour RTXP seul



(c) Délais pour RTXP avec R^{2}A (tous les messages)

FIGURE 6.10 – Délais des messages d'alarmes observés lors des simulations

l'échéance, car la capacité de RTXP est dépassée (cf. section 5.6 p.94). Cependant, le plus important pour une application critique de détection d'événement est que le premier message soit reçu avant l'échéance car on veut connaître la présence ou non de l'événement (comme pour le cas de la détection d'incendie par exemple). Avec la Figure 6.10(b) on constate que, pour tous les événements simulés, le premier paquet reçu par le puits dans le cas de RTXP seul respecte bien l'échéance. Cependant, après la réception de ce premier paquet, du fait des nombreux paquets qui restent à écouler, le réseau est inutilisable pour une durée qui dépend du nombre d'alarmes envoyées donc du nombre de nœuds qui ont détecté l'événement. Le cas de RTXP avec R^{2}A est représenté par la Figure 6.10(c). On constate que le nombre de paquets envoyés est bien moins important que dans le cas de RTXP seul (Figure 6.10(a)).

Tous les paquets respectent l'échéance. L'utilisation de R^{2}A permet donc la remonter d'alarmes en temps-réel sans saturation d'une partie du réseau.

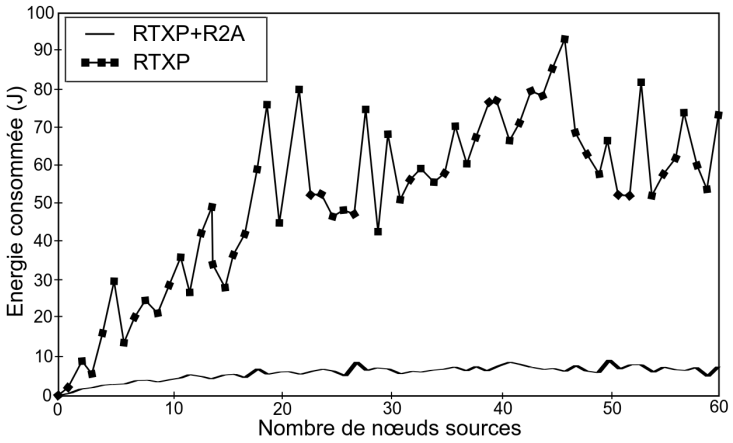


FIGURE 6.11 – Energie consommée durant les simulations

La Figure 6.11 représente l'énergie consommée par RTXP seul et RTXP avec R^{2}A durant les simulations (pour chaque nombre de sources on prend la valeur maximale d'énergie consommée observée). On constate que l'utilisation de R^{2}A réduit la consommation d'énergie d'un facteur 8 en moyenne. C'est dû au fait que beaucoup moins d'alarmes sont remontées au puits grâce à R^{2}A.

6.4 Conclusions sur R^{2}A

Dans ce chapitre nous proposons de réduire le nombre d'alarmes redondantes remontées vers le puits lors de la détection d'un événement. Nous proposons pour cela R^{2}A, un mécanisme d'agrégation basé sur les coordonnées virtuelles présentées dans le Chapitre 4. Ce mécanisme s'exécute durant les phases de sommeil de RTXP, cela permet de ne pas augmenter les délais de bout en bout de RTXP. R^{2}A effectue deux opérations : la synchronisation des nœuds qui détectent l'événement et la sélection de l'émetteur. Ces mécanismes sont réalisés à l'aide de messages non-explicites (*jamming codes*) ce qui rend cette approche très fiable [Boano et al., 2012]. Les résultats de simulations montrent que R^{2}A, en réduisant le nombre d'alarmes permet d'éviter une saturation du réseau (qui le rendrait sinon inutilisable pendant une durée non négligeable). Cette réduction du nombre d'alarmes permet de réduire considérablement (d'un facteur 8) l'énergie consommée par RTXP, ce qui augmente la durée de vie du réseau.

RTXP et R^{2}A fournissent donc avec le système de coordonnées du Chapitre 4 une solution complète de remontées d'alarmes en temps-réel pour les RCsF. Cette solution, en plus de fournir des garanties temporelles, est à la fois localisée, efficace en énergie, et adaptable aux conditions de trafic.

6.5 Synthèse de la première partie

Durant la première partie de ce document, nous nous sommes attachés à mettre évidence les problématiques clefs des transmissions temps-réel dans les RCsF et à proposer des mécanismes qui permettent de répondre à ces problématiques.

Les éléments qui sont nécessaires à l'établissement de communications temps-réel dans les RCsF sont :

1. des accès au canal déterministes pour pouvoir prédire le temps d'accès ;
2. un nombre de sauts pour atteindre le puits borné pour pouvoir borner le délai de bout en bout ;
3. une haute fiabilité pour répondre aux besoins des applications critiques ;
4. des solutions qui s'adaptent au trafic pour réduire la consommation d'énergie ;
5. l'utilisation de mécanismes locaux pour passer l'échelle.

Dans le Chapitre 2, nous montrons que les solutions proposées dans la littérature se focalisent seulement sur une partie des points listés ci-dessus : soit sur les aspects temps-réel (points 1 et 2) soit sur les aspects RCsF (points 3, 4 et 5), mais aucune ne prend en compte tous ces aspects.

Dans le Chapitre 3, nous étudions la fiabilité des RCsF, qui est définie comme la probabilité de succès d'une transmission de bout en bout, et déduisons que les solutions de routage opportuniste fournissent une fiabilité plus élevée que les solutions de routage classiques, nous montrons également que l'utilisation de plusieurs puits permet d'augmenter la fiabilité des transmissions pour les solutions opportunistes. Dans le Chapitre 4, nous proposons et évaluons un système de coordonnées virtuelles qui permet d'organiser le réseau pour pouvoir réaliser de manière déterministe des accès au canal et du routage. Dans le Chapitre 5, nous proposons RTXP, un protocole temps-réel *cross-layer* MAC et routage, qui permet de répondre aux problématiques listées ci-dessus. Il se base sur les résultats des chapitres précédents : il utilise un routage opportuniste pour augmenter la fiabilité et le système de coordonnées virtuelles pour organiser les accès au canal et router de manière déterministe. Enfin dans le Chapitre 6, nous proposons de résoudre le problème de tempête d'alarmes grâce à R^2A, un protocole d'agrégation qui réduit la redondance des informations envoyées au puits et permet ainsi de limiter la consommation d'énergie, la congestion et les collisions.

Nos travaux montrent qu'il est possible de réaliser des communications temps-réel fiables tout en prenant en compte les contraintes spécifiques liées aux RCsF (énergie, capacité, échelle).

Toutes nos propositions sont évaluées à l'aide de modèles théoriques et/ou de simulations. Les simulations, bien quelles soient utiles pour comparer les performances de nos propositions à celles de la littérature, permettent seulement d'explorer une partie des comportements possibles des systèmes étudiés, il n'est donc pas possible de conclure quant à l'absence d'erreurs. Dans la seconde partie de ce document, nous nous attachons à appliquer des techniques de vérification formelle aux protocoles temps-réel pour les RCsF, ces méthodes permettent de faire la preuve du bon fonctionnement

des protocoles vérifiés et donc de fournir les garanties requises par les applications critiques.

Deuxième partie

Vérification formelle de protocoles de communications temps-réel pour les réseaux de capteurs sans fil

Résumé de la deuxième partie

Dans la première partie de ce document, nous avons décrit les difficultés que représentent l'acheminement de données en temps-réel dans les RCsF. Nous avons évoqué les limitations des protocoles de la littérature (non garantie des bornes temporelles ou non prise en compte des ressources limitées des RCsF) et proposé une solution qui permet de lever ces limitations. La solution proposée permet de remonter des alarmes (qui correspondent à la détection d'un événement) en temps borné dans les RCsF.

Nous avons évalué les performances (délai, taux de livraison, etc) de notre proposition par simulation, cela nous permet d'en évaluer les comportements moyens et de comparer les performances de notre solution avec celles des protocoles de la littérature. Cependant, les applications critiques nécessitent un niveau de confiance très élevé : dans certains cas, des vies humaines dépendent du bon fonctionnement d'une application. Les simulations, bien que très utiles pour déceler certaines erreurs de conception qui induisent le non respect des spécifications du système, ne suffisent pas à assurer que le système a un fonctionnement correct (pas de blocage, respect des échéances, etc). L'utilisation de méthodes de vérification formelle, en plus des simulations et tests, permet d'augmenter le niveau de confiance dans le système [Baier and Katoen, 2008] en établissant des preuves de respect des spécifications.

Dans cette seconde partie, nous nous intéressons à l'application de méthodes formelles pour la vérification de propriétés temporelles des protocoles de RCsF. Dans la littérature, les méthodes formelles ont été largement utilisées pour la vérification de circuits électroniques, d'algorithmes, de protocoles réseau, etc [Baier and Katoen, 2008] [Clarke and Wing, 1996]. Cependant, l'application de ces techniques aux RCsF pose des problèmes au niveau de la modélisation du lien radio non-fiable et du passage à l'échelle [Kwiatkowska et al., 2002b] [Fruth, 2006] [Olveczky and Thorvaldsen, 2006] [Fehnker et al., 2007] [Singh et al., 2010]. En effet, les modèles utilisés ne représentent pas l'aspect probabiliste du lien radio et les méthodes de vérification utilisées ne permettent que de vérifier des protocoles sur des réseaux de quelques nœuds (moins de 10 dans la plupart des cas [Kwiatkowska et al., 2002b] [Watteyne et al., 2006a] [Olveczky and Thorvaldsen, 2006] [Fehnker et al., 2007]).

Dans le Chapitre 7, nous présentons les méthodes, formalismes de modélisation et outils disponibles pour la vérification formelle des RCsF. Nous mettons en avant les différents aspects des RCsF à modéliser et choisissons un formalisme et un outil qui leur sont adaptés. Dans le Chapitre 8, nous présentons les spécificités des RCsF : lien radio qui induit des transmissions broadcast *et qui est non-fiable, et la grande taille de ce type de réseaux, qui rendent difficile leur vérification. Nous proposons une méthodologie de vérification temporelle de protocoles pour RCsF sous forme de trois contributions qui répondent à ces difficultés :*

1. *Une étude de la modélisation de la nature* broadcast *des transmissions sans fil. Cette étude porte sur les modélisations rencontrées dans la littérature et permet de sélectionner celle qui permet un meilleur passage à l'échelle lors de la vérification.*

2. *Une technique de modélisation des liens radios non-fiables pour la vérification des protocoles de RCsF.*
3. *Une technique qui permet de faire face à la grande taille potentielle des réseaux vérifiés.*

Dans le Chapitre 9, nous présentons les résultats de l'application de cette méthodologie de vérification à RTXP, proposé dans le Chapitre 5 de la première partie de ce document.

Chapitre 7

Techniques de vérification formelle pour le temps-réel dans les RCsF

La vérification formelle consiste en l'application de méthodes formelles : une méthode formelle est une technique qui permet de démontrer de façon rigoureuse qu'un système respecte un ensemble de propriétés [Clarke and Wing, 1996]. Pour cela, le système et les propriétés sont modélisés dans un langage mathématique (ou formalisme). Ce langage permet de raisonner sans ambiguïtés sur le système pour établir la véracité ou non des propriétés étudiées : on vérifie donc un modèle du système. Pour vérifier de manière formelle un système, il faut donc trois éléments :

1. une méthode de vérification : la façon d'établir la preuve de bon fonctionnement ;
2. un formalisme : le langage de modélisation du système ;
3. un outil : qui nous permet de modéliser le système étudié dans le formalisme choisi et qui implémente la méthode de vérification.

Dans le cadre des protocoles temps-réel pour RCsF, nous nous intéressons particulièrement au pire cas du délai de bout en bout : il faut que cette valeur soit bornée. Dans ce chapitre, nous décrivons les méthodes, formalismes et outils existants et nous déterminons lesquels sont à même de permettre la modélisation et vérification des protocoles temps-réel pour RCsF. Nous décrivons ensuite des applications de ces techniques rencontrées dans la littérature et montrons leurs limitations (non passage à l'échelle, absence de modélisation du lien non-fiable).

7.1 Méthodes formelles

Dans cette section, nous nous intéressons aux méthodes qui permettent d'établir formellement que le modèle d'un système respecte ses spécifications.

7.1.1 Validation et vérification

Avant de présenter les différentes techniques existantes, il nous semble important de préciser les termes de validation et de vérification, car dans la littérature leurs

significations varient (et sont même parfois interchangeables). Dans ce document, nous les définissons de la manière suivante [IEEE, 2011a] :

Définition 7.1.1 *La **validation** est le processus qui permet d'assurer que le système répond bien aux attentes du maître d'ouvrage. La **validation** s'applique au processus de développement dans son ensemble pour être sûr que le résultat de ce développement est le produit attendu.*

Définition 7.1.2 *La **vérification** permet de s'assurer qu'un système respecte une série de spécifications. Ce qui différencie ce processus de celui de la validation est le fait que les spécifications vérifiées, si elles ne sont pas validées, peuvent ne pas correspondre aux attentes du maître d'ouvrage. La vérification formelle d'un système nécessite sa modélisation en langage mathématique.*

On distingue, dans la littérature scientifique, deux méthodes de vérification formelle généralistes : la preuve et le *Model Checking* [Clarke and Wing, 1996]. Il existe aussi une méthode plus spécifiquement orientée pour la vérification de bornes temporelles dans les réseaux : le *Network Calculus* [Le Boudec and Thiran, 2001]. Nous les présentons dans les sections suivantes.

7.1.2 Preuve

Avec la preuve (ou preuve de théorème), le système et les propriétés sont modélisés grâce à des formules de logique mathématique. La logique comporte des axiomes (qui sont admis) et des règles d'inférences qui permettent de produire les théorèmes. La réalisation de preuve consiste donc à appliquer les règles d'inférences sur le modèle du système. Ce processus peut être réalisé à la main ou bien de manière semi-automatique grâce à des logiciels assistants de preuve (Coq [Coq, 2010], HOL [Slind and Norrish, 2008], Isabelle [Isabelle, 2011], etc).

La preuve formelle permet de vérifier des systèmes qui ont un nombre d'états infini (par exemple avec des variables qui peuvent prendre une infinité de valeurs). Cependant, ce type de méthodes, non automatisées, demande une grande expertise de la part de l'utilisateur. De plus, le processus de preuve, souvent long, laisse la place à l'introduction d'erreurs subtiles [Clarke and Wing, 1996].

7.1.3 *Model Checking*

Le *Model Checking* est une méthode de vérification formelle automatique. Le système étudié est décrit sous forme de modèle à états et transitions qui représente ses spécifications. A partir de ce premier modèle est généré un second modèle qui, comme on le verra dans la section 7.2.1, est une interprétation sémantique du premier et qui représente tous les comportements possibles du système, c'est-à-dire ses exécutions possibles. Des algorithmes permettent l'exploration exhaustive de l'espace d'états des comportements [Baier and Katoen, 2008]. Durant cette exploration, certains types de propriétés sont vérifiées : accessibilité (il existe un chemin d'exécution vers un état

donné), sûreté (vrai quel que soit l'état), blocage (un état n'a pas de transition sortante), etc. Par exemple dans le cas d'une vérification de protocole temps-réel, on vérifiera qu'aucun état dans lequel un paquet a un délai de bout en bout supérieur à l'échéance n'est accessible.

Les principaux avantages du *Model Checking* sont :

- L'exhaustivité de la vérification : absolument tous les comportements du modèle sont explorés, cela permet de détecter toutes les violations de spécification.
- Le fait que la méthode soit automatique : cela évite les erreurs humaines évoquées dans le cas des preuves et permet de se concentrer sur la qualité de la modélisation.
- La capacité à fournir des contre-exemples : lorsqu'une propriété n'est pas vérifiée, le *Model Checker* est capable de fournir une trace qui permet au concepteur du système de comprendre le problème.

En revanche, la principale limitation du *Model Checking* est l'explosion combinatoire. En effet, lorsque le système est complexe, distribué et/ou large échelle, le nombre de comportements possibles devient très important. Il est alors impossible de stocker ou d'explorer en un temps raisonnable le modèle des comportements du système, ce qui rend la vérification infaisable en pratique.

7.1.4 Méthode analytique : le *Network Calculus*

Le *Network calculus* [Le Boudec and Thiran, 2001] est un *framework* théorique pour analyser les performances des réseaux de communication. Ce formalisme donne des bornes strictes sur les performances du système. Les communications constituent des flux qui s'écoulent dans le réseau. Des contraintes sur ces flux sont définies sous forme de fonctions mathématiques appelées courbes d'arrivées (qui contraignent le trafic entrant dans les nœuds) et courbes de service (qui définissent le service minimum du trafic qui entre dans les nœuds). Ces fonctions peuvent êtres composées en utilisant des opérateurs de l'algèbre min-plus [Le Boudec and Thiran, 2001].

Ce type de méthodes permet d'évaluer les bornes théoriques de systèmes large échelle (les calculs consistent à exécuter des fonctions maximum, minimum et des sommes [Wandeler, 2006] qui sont peu gourmandes en ressources). Cependant, la modélisation d'un protocole de communication sous forme de courbe de service se fait en se basant sur des hypothèses non vérifiées : on suppose que le nœud est capable de fournir au moins un certain service, sans en faire la preuve (pour des protocoles simples, c'est évident - le TDMA par exemple - pour des protocoles aux comportements plus complexes - comme RTXP par exemple - le risque d'erreur est plus grand). De plus, l'architecture du réseau est donnée comme un paramètre d'entrée, le calcul des bornes se fait pour une topologie donnée, sans possibilité de représenter le dynamisme de la topologie (disparitions/apparitions de nœuds / de liens).

7.1.5 Quelle méthode pour vérifier des protocoles de RCsF ?

Le but de cette deuxième partie du document est de fournir une méthodologie de vérification de protocoles temps-réel pour les RCsF. Pour le choix de la méthode formelle, nous considérons les critères suivants :

- Son applicabilité au problème qui nous intéresse : il faut que la méthode nous permette d'analyser les comportements temporels des RCsF.
- Le niveau de confiance accordé à la technique.

[Fehnker et al., 2007], [Singh et al., 2010] et [Schmitt and Roedig, 2005] montrent que les trois méthodes précédemment présentées sont applicables aux réseaux sans fil. Dans les trois cas, les méthodes permettent d'analyser les comportements temporels des protocoles de communication. Notre critère de sélection principal est donc le niveau de confiance que l'on peut accorder à chaque méthode.

Le *Network Calculus* est spécifiquement développé pour étudier les cas pires de délais de bout en bout dans les réseaux. Cependant, il nous paraît difficile d'établir la correspondance évoquée entre protocoles et courbes de service de façon sûre, en particulier pour des protocoles au comportement complexe. En effet, un protocole de type TDMA est aisément représentable en *Network Calculus* [Le Boudec and Thiran, 2001] mais il n'est pas évident que ce soit le cas des protocoles distribués large échelle des RCsF.

Les méthodes de preuve et de *Model Checking*, quant à elles, ne sont pas spécifiquement prévues pour vérifier les bornes temporelles des protocoles de communication. Cependant ces méthodes ont été utilisées à de nombreuses reprises pour cela dans la littérature [Clarke and Wing, 1996].

La preuve, bien qu'elle permette d'atteindre un niveau de confiance équivalent au *Model Checking* quand elle est réalisée sans erreur, doit être effectuée par un expert ayant une grande connaissance du système formel utilisé [Godary, 2004]. De plus, le processus de preuve n'est pas automatisé, cela le rend plus vulnérable à l'erreur humaine que le *Model Checking* [Clarke and Wing, 1996].

Nous nous orientons vers le *Model Checking* du fait que le processus soit automatique (laissant moins de place à l'erreur humaine durant la vérification que pour la preuve) et qu'il fournisse, grâce à l'exploration exhaustive de tous les comportements du système, des contre-exemples très utiles dans le processus de développement d'un protocole. C'est donc la méthode qui, de notre point de vue, offre le plus haut niveau de confiance parmi trois méthodes présentées. Cependant, le *Model Checking* souffre du problème d'explosion combinatoire. Nous verrons que cet aspect est limitant pour vérifier les RCsF dont une des principales caractéristiques est d'être large échelle. Une partie de nos contributions porte sur ce problème.

Nous n'écartons cependant pas totalement le *Network Calculus*. En effet, les travaux de [Lampka et al., 2009] sur les relations entre courbes d'arrivée et de service et modèles à états-transitions laissent penser qu'il est possible de vérifier formellement, par *Model Checking*, la correspondance d'une courbe de service à un protocole donné. Cette piste est explorée dans le chapitre 8, où nous tirons parti du passage à l'échelle de *Network Calculus* et de l'exhaustivité du *Model Checking*.

De même, l'utilisation de la preuve peut être envisagée conjointement avec celle du *Model Checking*. Par exemple, pour réaliser des abstractions du modèle avant de le vérifier [Godary, 2004]. Nous n'explorerons cependant pas cette possibilité dans ce document.

Pour vérifier formellement un système, il faut d'abord pouvoir le modéliser. Dans la section suivante nous énumérons les aspects des RCsF qu'il faut modéliser et évaluons si les différents formalismes existants sont capables de représenter ces aspects.

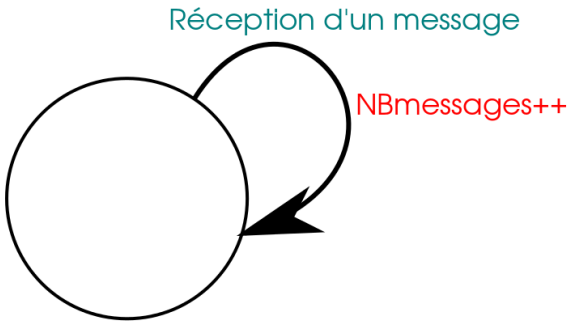
7.2 Formalismes de modélisation

Dans cette section, nous nous intéressons aux différents aspects des RCsF à modéliser et aux formalismes existants pour les modéliser formellement. Nous nous intéressons aux formalismes qui permettent de réaliser ensuite une vérification par *Model Checking*. Avant de détailler les aspects des RCsF et les formalismes, nous établissons la différence entre le modèle d'un système et son interprétation sémantique.

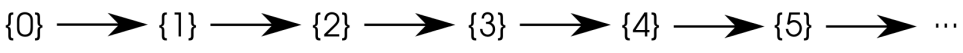
7.2.1 Modèle du système et interprétation sémantique

Pour vérifier formellement un système par *Model Checking*, l'utilisateur fournit un modèle du système qui en décrit les règles de comportement. Par exemple pour un système de communication on peut imaginer une règle du type "Quand le nœud reçoit un message, il incrémente sa variable *NBmessages*". Ces règles sont décrites dans un formalisme de modélisation (nous en détaillerons plusieurs dans les sections suivantes) qui est capable de représenter les différents aspects des RCsF (également présentés dans les sections suivantes). Le processus de *Model Mhecking* consiste à explorer toutes les exécutions possibles du système, c'est-à-dire toutes les interprétations possibles des règles de comportement à partir d'un état initial de ses variables. L'interprétation du modèle (appelée interprétation sémantique) est représentée sous forme d'un graphe d'états (noté LTS pour *Labeled Transition System*) où chaque état est un enregistrement des valeurs des variables du système [Baier and Katoen, 2008]. Les états sont reliés entre eux par l'application des règles de comportement.

La Figure 7.1(a) illustre un modèle qui correspond à la règle "Quand le nœud reçoit un message, il incrémente sa variable *NBmessages*" sous forme d'automate et la Figure 7.1(b) montre son interprétation sémantique sous forme de LTS. Dans ce cas un état du LTS correspond à la valeur de la seule variable du système, *NBmessages* qui est ici initialisée à 0. On peut noter que si la valeur de *NBmessages* n'a pas de borne supérieure, le LTS est infini : cela pose problème car l'exploration exhaustive du LTS est alors impossible. Même si le LTS est fini, la complexité du système modélisé (typiquement, un système avec un grand nombre de variables) engendre une explosion combinatoire du nombre d'états. Le LTS ne peut alors plus être stocké en mémoire, c'est le principal problème du *Model Checking*, on reviendra sur ce problème en détails dans le Chapitre 8. L'utilisateur doit donc tenir compte de ces problèmes lors de la modélisation du système [Baier and Katoen, 2008].



(a) Modèle



(b) Interprétation sémantique

FIGURE 7.1 – Modèle et interprétation sémantique d'un nœud qui incrémente une variable lorsqu'il reçoit un message

7.2.2 Modélisation des RCsF

Les RCsF sont des réseaux multi-sauts sans fil large échelle. Pour modéliser ce type de système il faut pouvoir représenter les aspects suivants :

- La concurrence : chaque nœud exécute des actions en parallèle des autres nœuds.
- Les communications : les nœuds voisins dans la topologie peuvent communiquer entre eux et interfèrent entre eux du fait de la nature *broadcast* du médium radio.
- Les données : il faut pouvoir disposer de structures de données pour représenter les informations du système (informations contenues dans les paquets, topologie du réseau, etc).
- Les probabilités : les liens radios sont non-fiables, donc les communications sont probabilistes, il faut donc pouvoir exprimer les probabilités de réception et de perte des paquets.
- Le temps : dans ce document nous nous intéressons aux comportements temporels des protocoles pour RCsF.

Ces aspects doivent donc être représentables dans le formalisme utilisé pour modéliser les RCsF. Le dernier aspect, le temps, peut être modélisé de deux façons différentes dans les représentations à états et transitions. Nous nous intéressons à cet aspect dans la section suivante.

7.2.3 Représentation du temps

Dans ce document, nous nous intéressons aux comportements temporels des protocoles pour RCsF. Nous voulons notamment vérifier que les paquets respectent une échéance. Pour cela il est nécessaire d'avoir une représentation quantitative (explicite) du temps, c'est-à-dire que le modèle doit permettre de mesurer le temps entre deux événements (typiquement, l'émission et la réception d'un paquet). Dans les LTS, le

temps peut être représenté de manière discrète ou continue. La représentation discrète du temps est réalisée en prenant comme durée élémentaire la transition : c'est-à-dire que le passage d'un état à un autre prend une unité de temps. La représentation d'un délai se fait par une succession de transitions dîtes silencieuses. Cette représentation peut induire des LTS très grands dans le cas où les échelles de temps des composants du système sont différentes [Baier and Katoen, 2008]. Cependant, ce type de représentation fonctionne bien pour les systèmes à petite échelle et synchrones tels que les circuits électroniques.

En revanche, pour les systèmes asynchrones à large échelle avec des contraintes temporelles variées tel que les RCsF, il est préférable de disposer d'une représentation explicite et continue du temps [Baier and Katoen, 2008]. Dans ce cas, les comportements du système sont représentés en ajoutant une date ($\in \mathbb{R}_+$) aux transitions du LTS qui devient alors un TLTS (Timed LTS) [Geilen, 2002].

Nous nous orientons donc vers un formalisme avec une représentation quantitative et explicite du temps. Dans les sections suivantes, nous présentons plusieurs formalismes possibles.

7.2.4 Les réseaux de Petri

Les réseaux de Petri sont historiquement le premier formalisme de modélisation de la concurrence [Baeten, 2005].

Formellement un réseau de Petri est un tuple $PN = (P, T, F, W, M_0)$ avec :

- $P = \{p_0, p_1, ..., p_m\}$ un ensemble de places (qui sont marquées avec des jetons)
- $T = \{t_0, t_1, ..., t_n\}$ un ensemble de transitions
- $F \subseteq (P \times T) \cup (T \times P)$ un ensemble d'arcs orientés qui relient des places à des transitions et des transitions à des places
- $W : F \rightarrow (1, 2, 3, ...)$ une fonction qui assigne un poids à chaque arc (par exemple : $w(p_3, t_4) = 2$)
- $M_0 : P \rightarrow (0, 1, 2, ...)$ qui assigne un marquage initial aux places : le marquage est un entier associé à une place, il correspond au nombre de jetons contenus dans les places.

Le comportement du système est décrit par l'évolution du marquage des places, cette évolution dépend du marquage initial et du franchissement des transitions. Une transition t est dite active si la place p_i est marquée avec au moins $w(p_i, t)$ jetons. Une transition parmi les transitions actives est choisie de manière non-déterministe pour être exécutée. Quand c'est le cas, $w(p_i, t)$ jetons sont retirés de la place de départ p_i et $w(t, p_o)$ jetons sont ajoutés à la place d'arrivée p_o.

Une extension temporelle des réseaux de Petri permet une représentation explicite du temps [Merlin and Farber, 1976]. Les réseaux de Petri temporels ajoutent des contraintes de temps sur le franchissement des transitions. Ces contraintes ont la forme d'intervalles de temps durant lesquels il est possible de prendre la transition.

Formellement, un réseau de Petri temporel est un tuple $TPN = (P, T, F, W, M_0, I)$ où P, T, F, W et M_0 ont les mêmes définitions que pour les réseaux de Petri classiques et où :

- $I : T \rightarrow Q^+ \times Q^+$ assigne un intervalle de temps à chaque transition. Une transition peut être prise seulement durant l'intervalle.

L'interprétation sémantique d'un réseau de Petri temporel est représentable par un TLTS (Timed LTS). Le *Model Checking* décidable sur un TLTS [Baier and Katoen, 2008].

En revanche, les réseaux de Petri temporels ne permettent pas d'associer des variables (ou structures) de données au modèle, on peut cependant modéliser des compteurs ou booléens à l'aide du marquage des places. Les communications entre nœuds sont représentables en reliant deux modèles de nœud par une place partagée [Godary, 2004]. Pour modéliser un réseau sans fil, cela impose de représenter toute la topologie explicitement dans le modèle (on ne peut pas stocker la topologie dans une structure de donnée). Cette approche est contraignante pour représenter des systèmes large échelle tels que les RCsF. Enfin, il existe une version temporelle stochastique [Atamna, 1994] des réseaux de Petri qui permet de représenter à la fois le temps et les probabilités, mais à notre connaissance, aucun outil capable de vérifier ce formalisme n'est disponible.

7.2.5 Les algèbres de processus

Une algèbre de processus permet de modéliser les comportements des systèmes distribués par des expressions mathématiques. Une algèbre de processus est une structure mathématique dont les opérateurs satisfont une série d'axiomes, un processus étant un élément de l'algèbre. On peut citer comme exemples d'algèbres de processus CCS [Milner, 1989], CSP [Hoare, 1985], ACP [Bergstra and Klop, 1984], etc.

Dans le cas de CCS, pour un ensemble d'actions données, les processus sont définis de la manière suivante :

- $\emptyset$ est le processus vide
- $a.P_1$ exécute une action a et continue le processus P_1
- $A = P_1$ A est utilisé comme identifiant du processus P_1
- $P_1 + P_2$ représente le non-déterminisme, le processus peut se comporter comme P_1 ou bien P_2
- $P_1|P_2$ est la composition parallèle, cela signifie que les deux processus s'exécutent indépendamment
- $P_1[b/a]$ renomme l'action a en b pour le processus P_1
- $P_1 \backslash a$ se comporte comme P_1 sans l'action a

Les axiomes de l'algèbre sont définis comme suit :

- $P_1 + P_2 = P_2 + P_1$
- $P_1 + (P_2 + P_3) = (P_1 + P_2) + P_3$
- $P_1 + P_1 = P_1$
- $(P_1 + P_2).P_3 = P_1.P_3 + P_2.P_3$
- $P_1|P_2 = P_2|P_1$
- $(P_1|P_2)|P_3 = P_1|(P_2|P_3)$

Le système étudié est représenté sous forme de processus qui communiquent entre eux. Les communications sont des synchronisations entre deux processus sur une action. La concurrence et les communications sont donc nativement représentables avec

les algèbres de processus. En revanche, ce n'est pas le cas des données et des structures de données mais elles peuvent tout de même être aisément ajoutées [Milner, 1993] [Abadi and Fournet, 2001].

La sémantique de l'algèbre de processus est un LTS où les états sont des expressions de processus et les transitions correspondent à des actions. Les règles pour la construction du LTS à partir du modèle du système sont énoncées en détail dans [Milner, 1989] pour le cas du CCS.

Une version du CCS propose la représentation explicite du temps [Wang, 1990]. La définition de l'algèbre est étendue avec un opérateur de délai :

- $\delta_\tau.P_1$ attend δ_τ et continue P_1.

La sémantique de cette extension de CCS est un TLTS [Wang, 1990].

Il existe aussi une extension probabiliste de CCS, PCCS [Jou and Smolka, 1990] qui remplace les comportements non-déterministes par des comportements probabilistes :

- $P_1 + P_2$ est remplacé par $\sum_{i \in I}[p_i]P_i$ où $p_i \in (0, 1]$ et $\sum_{i \in I} p_i = 1$

Une algèbre de processus qui combine comportements probabilistes et temporels est proposée dans [Hansson and Jonsson, 1990], mais, à notre connaissance, aucun outil capable de vérifier ce formalisme n'est disponible. De plus, [Singh et al., 2010] propose une algèbre de processus pour les réseaux mobiles ad hoc. Le formalisme permet de décrire la nature *broadcast* du sans fil car les synchronisations ne se font plus entre deux nœuds seulement mais dans un groupe de nœuds. Les groupes peuvent évoluer au cours de l'exécution du modèle, cela permet de représenter des changements de topologie. Cependant, cette algèbre de processus ne propose pas de représentation explicite du temps et ne permet pas de représenter des comportements probabilistes.

Les algèbres de processus présentés ci-dessus permettent donc, si on les considère dans leur ensemble, de représenter tous les aspects des RCsF. Cependant, pris un par un, ils n'en sont pas capables : [Hansson and Jonsson, 1990] qui permet de représenter à la fois le temps et les probabilités, ne permet pas de représenter aisément la nature *broadcast* du sans fil et [Singh et al., 2010] qui permet de représenter le *broadcast* et la dynamique de la topologie, ne permet pas de représenter les probabilités et le temps. De plus, il n'existe pas à notre connaissance d'outils de *Model Checking* pour ces algèbres (bien qu'il en existe de nombreux pour d'autres algèbres [Cleaveland and Sims, 2002], [Garavel et al., 2013], [Sun et al., 2009]).

7.2.6 Les automates

Le concept d'automate est assez large, on peut le définir comme une machine comportant des entrées, des sorties et des états internes. En fonctions des entrées et des états internes, des sorties sont produites. Cette description générale est à la base de nombreux formalismes [Baier and Katoen, 2008] utilisés pour la modélisation théorique des langages et de la complexité des algorithmes [Papadimitriou, 2003]. Dans cette section nous nous concentrons sur les automates temporisés [Alur and Dill, 1994] et leurs variantes qui permettent de modéliser des systèmes temps-réel grâce à la représentation explicite du temps. Nous présentons ici la version TSA (*Timed Safety Automata*) des automates temporisés [Henzinger et al., 1992] qui est celle utilisée dans

la littérature pour la modélisation des systèmes temps-réel. Dans la suite de ce document nous ferons référence aux TSA avec l'abréviation TA (Timed Automata) comme c'est l'usage dans la littérature [Baier and Katoen, 2008].

Formellement, un TA est un tuple $TA = (Loc, Act, X, \rightarrow, Loc_0, Inv, L)$ où :

- Loc un ensemble d'états (à ne pas confondre avec les états du LTS, c.f. section 7.2.1) $Loc_0 \subseteq Loc$ est un ensemble d'états initiaux ;
- Act est un ensemble d'actions ;
- X est un ensemble d'horloges ;
- $\rightarrow \subseteq Loc \times \zeta(X) \times Act \times 2^X \times Loc$ est la relation de transition avec $\zeta(X)$ une contrainte d'horloge nommée garde (par exemple $x < 3 \wedge y < 4$ avec $x, y \in X$) ;
- $Inv : Loc \rightarrow \zeta(X)$ est la fonction d'assignation des invariants (qui sont des contraintes d'horloge) aux états ;
- $L : Loc \rightarrow 2^{AP}$ est une fonction d'étiquetage des états avec AP un ensemble de propositions atomiques.

Dans un état, le temps peut s'écouler tant que l'invariant est vrai. Toute transition active depuis cet état peut être prise, une transition est active si la contrainte sur les horloges (nommée garde) est vérifiée. La transition est sélectionnée de manière non déterministe parmi les transitions actives. Quand la transition est prise, un sous-ensemble des horloges (2^X) est remis à 0.

L'interprétation sémantique d'un TA est un TLTS pour lequel les états (que nous nommons configurations par la suite pour éviter toute ambiguïté par rapport aux états du TA) sont définis par le tuple (l, v) où $l \in Loc$ est un état du TA et $v \in \nu(X)$ un ensemble de valeurs des horloges de X [Baier and Katoen, 2008]. Le *Model Checking* est décidable sur cette structure [Baier and Katoen, 2008].

Les TA permettent de représenter le comportement temporel des systèmes, en revanche il est plus difficile de représenter les communications et la concurrence car il faudrait encoder la composition parallèle ainsi que les synchronisations de plusieurs éléments du système directement avec un TA. Du fait du nombre de possibilités d'entrelacements entre les actions des différents éléments (l'augmentation est quadratique avec le nombre d'éléments), cet encodage deviendrait rapidement impossible humainement lors de l'augmentation de la taille du système. Par conséquent, pour représenter la concurrence et les communications, les réseaux de TA (NTA - *Network of TA*) ont été introduits [Olderog and Dierks, 2008]. Un NTA est la composition parallèle de plusieurs TA qui ont la possibilité de se synchroniser sur des actions communes (chaque outil de *Model Checking* utilise un style de composition et de synchronisation propre emprunté à un algèbre de processus : par exemple UPPAAL [Gerd Behrmann and Larsen, 2004] utilise les opérateurs de CCS). Cela permet de définir le comportement de chaque élément du système par un TA qui est ensuite composé avec les autres. Formellement, un NTA est la composition parallèle de $\{A_i\}_{1 \leq i \leq n}$ TA où $A_i = (Loc_i, Act, X, \rightarrow_i, Loc_i^0, Inv_i, L_i)$ pour $1 \leq i \leq n$. La notion de configuration du TLTS associé devient alors le tuple $(\bar{l}, v)$ où $\bar{l} \in L_1 \times ... \times L_n$ est un vecteur d'états et $v \in \nu(X)$ un ensemble de valeurs des horloges de X. Le *Model Checking* reste décidable sur cette structure [Baier and Katoen, 2008].

Les NTA offrent la possibilité de modéliser le temps, la concurrence, les communications (par synchronisation). Il est possible d'y ajouter la gestion des données [Olderog and Dierks, 2008]. En effet, on modifie pour cela la définition du TA :

- On ajoute V un ensemble de variables de données.
- La relation de transition devient $\rightarrow \subseteq Loc \times \zeta(X) \times \zeta(V) \times Act \times 2^X \times up(2^V) \times Loc$, en plus des contraintes sur les horloges, on ajoute des contraintes sur les variables de données, $\zeta(V)$. On ajoute aussi des mises à jour des valeurs des variables, $up(2^V)$.

Il est en plus possible d'ajouter des transitions probabilistes aux TA qui deviennent alors de PTA (*Probabilistic Timed Automata*) :

- la transition devient alors $\rightarrow_{prob} \subseteq Loc \times \zeta(X) \times \zeta(V) \times Act \times Dist(2^X \times up(2^V) \times Loc)$ une transition probabiliste.

Dans ce cas, la transition est choisie parmi les transitions actives, puis l'état d'arrivé (associé à des remises à zéro d'horloges et des mises à jour de variables) est choisi en fonction d'une distribution de probabilités. De la même façon que les TA, les PTA sont composables en NPTA (Network of PTA) et peuvent partager des variables de synchronisation. La sémantique d'un NPTA est un MDP (*Markov Decision Process* - Processus de décision Markovien) [Kwiatkowska et al., 2006] sur lequel le *Model Checking* est décidable.

Les PTA permettent donc de modéliser tous les aspects évoqués dans la section 7.2.2. De plus il est possible de réaliser une vérification par *Model Checking* de ce formalisme. Nous orientons donc notre choix vers ce formalisme pour la modélisation des RCsF.

7.2.7 Modélisation des propriétés à vérifier

Le principe du *Model Checking* est de vérifier qu'une ou plusieurs propriétés sont respectées par un modèle du système. Dans les sections précédentes, nous nous sommes intéressés aux formalismes pour la modélisation du système, nous nous tournons maintenant vers ceux dédiés à l'expression des propriétés à vérifier. Les propriétés correspondent aux spécifications du système (par exemple "Le système n'entre jamais dans un état de blocage" ou "Avec une probabilité d'au moins 99%, la latence maximale d'un paquet est toujours en dessous de 5 secondes" sont des propriétés). Il est à noter que dans cette section nous raisonnons sur les exécutions possibles du système, donc sur l'interprétation sémantique du modèle (le LTS). Les propriétés peuvent être classées par type [Baier and Katoen, 2008], [Haddad, 2011] :

- **L'accessibilité** : permet de vérifier qu'une configuration est atteignable (par une suite de transitions) depuis une autre configuration du système. Ce type de propriété nous intéresse particulièrement, car il permet de vérifier la sûreté du système : on peut vérifier qu'aucune configuration "mauvaise" n'est accessible depuis la configuration initiale du système. Par exemple dans notre cas, on pourra vérifier que dans aucune configuration accessible, un paquet a un délai qui dépasse l'échéance. D'autres types de propriétés sont définis à partir de l'accessibilité.

- La **quasi-vivacité** s'applique aux actions du système (non aux configurations). Une action est quasi-vivante s'il existe deux configurations e et e' accessibles telles que la transition de e à e' déclenche cette action.
- La **vivacité** vérifie que pour toutes les configurations accessibles depuis la configuration initiale, l'action est quasi-vivante.
- **Configuration d'accueil** : vérifie qu'une configuration est accessible depuis toute configuration accessible depuis la configuration initiale.
- **Configuration inévitable** : vérifie que la configuration est une configuration d'accueil et qu'elle est visitée une infinité de fois durant une exécution infinie du système.

Ces types de propriétés pour pouvoir être vérifiées doivent être exprimés dans des logiques temporelles [Baier and Katoen, 2008]. Il en existe deux grandes catégories :

- LTL (Linear Timed Logic) qui raisonne sur des chemins d'exécution.
- CTL (Computational Tree Logic) qui raisonne sur des arbres d'exécution.

Ces deux logiques ont des dérivées pour prendre en compte explicitement le temps et les probabilités (PLTL, TLTL, PCTL, TCTL, PTCTL) [Baier and Katoen, 2008]. On peut noter que dans le cas des propriétés probabilistes on va s'intéresser au cas pires de probabilités d'atteindre une configuration, donc soit la probabilité maximum (d'arriver dans une "mauvaise" configuration), soit la probabilité minimum (d'être toujours dans une "bonne" configuration).

Du choix de l'une ou l'autre logique dépendent les algorithmes de *Model Checking* [Baier and Katoen, 2008]. Dans [Haddad, 2011], l'auteur montrent que leurs expressivités ne sont pas équivalentes. En pratique, le choix d'une ou de l'autre logique est imposé par l'outil de *Model Checking* et quel que soit le choix, la logique permet de vérifier la sûreté du système (non accessibilité d'une "mauvaise" configuration). Nous ne détaillerons donc pas plus ici les subtilités de ces logiques (plus d'information peut être trouvée dans [Baier and Katoen, 2008] et [Haddad, 2011]).

7.2.8 Conclusion sur le choix d'un formalisme de modélisation

Dans cette section, nous présentons les différents aspects des RCsF à modéliser et proposons différents formalismes qui pourraient convenir. Nous déterminons d'abord, dans la section 7.2.3, que pour vérifier des propriétés temporelles dans les RCsF il faut modéliser le temps de façon explicite et continu. Nous décrivons ensuite les formalismes qui permettent cette représentation. Le Tableaux 7.2.8 résume les capacités des formalismes décrits dans cette section, on constate que plusieurs pourraient convenir car ils permettent de représenter les aspects des RCsF énoncés dans la section 7.2.2. Les algèbres de processus et les automates permettent de représenter à la fois la concurrence, les communications, les données, les probabilités et le temps. Cependant, le manque d'outils de modélisation et vérification (qui réunissent tous ces aspects) pour les algèbres de processus oriente notre choix vers les automates (les PTA). Les PTA offrent la possibilité de modéliser tous les aspects des RCsF, le

	Réseaux de Petri	Algèbre de processus	Automates
Concurence	✓	✓	✓
Communications	✓	✓	✓
Données	✗	✓	✓
Probabilités	✓	✓	✓
Temps explicite et continu	✓	✓	✓
Outil†	✗	✗	✓

TABLE 7.1 – Résumé des capacités des formalismes de modélisation

Model Checking de ce formalisme est décidable et il existe des outils de modélisation et vérification qui seront présentés dans la section suivante.

7.3 Outils de *Model Checking*

Nous nous concentrons dans cette section sur les outils ayant un pouvoir d'analyse qui permet de vérifier à la fois des propriétés temporelles et probabilistes (par exemple, "Avec une probabilité d'au moins 99%, la latence maximale d'un paquet est toujours en dessous de 5 secondes") sur des PTA. Nous avons référencé trois outils qui possèdent ces caractéristiques : Fortuna [Berendsen et al., 2010], PRISM [Kwiatkowska et al., 2011] et UPPAAL [Gerd Behrmann and Larsen, 2004].

7.3.1 Fortuna

Fortuna est un *model checker* pour les PPTA (Priced PTA), qui sont des PTA étendus avec des coûts sur les transitions (le passage d'une transition a un coût) et des coûts par unité de temps dans les états du PTA. Les auteurs de [Berendsen et al., 2009] montrent que le *Model Checking* sur les PPTA n'est pas décidable dans le cas général. Cependant, il existe un semi-algorithme (algorithme dont la terminaison n'est pas garantie pour toutes les entrées) qui permet de calculer la probabilité maximale d'accéder à une configuration sous contrainte de coût et de temps. On peut noter que même si l'objectif de ce document est de valider des protocoles temps-réel avant tout, l'ajout de coût permet de modéliser la consommation énergétique qui est un paramètre essentiel dans les RCsF. On peut alors imaginer vérifier des propriétés du type : "Avec une probabilité de 99% un paquet arrive au puits en moins de 5 secondes et en dépensant moins de 10 unités d'énergie".

Fortuna est entièrement écrit en C++. Les modèles PPTA doivent aussi être encodés en C++. Malgré une étude [Berendsen et al., 2010] qui montre que les performances de Fortuna sont meilleures que celles de PRISM (pour les deux cas d'études évalués), Fortuna reste pour l'instant peu utilisé et pas documenté. Ce manque de maturité est un point bloquant, car le manque de retour d'expérience des utilisateurs ne nous permet pas de nous faire une idée sur la fiabilité de l'outil. De plus, l'absence,

†. Il s'agit ici d'un outil prenant en compte tous les aspects cités

à notre connaissance, de documentation rend toute tentative d'implémentation de modèle très délicate.

7.3.2 PRISM

PRISM est un *model checker* probabiliste qui permet de travailler avec des chaînes de Markov (temps discret et continu), des MDP et des PTA. PRISM propose deux algorithmes pour la vérification des PTA : le premier basé sur la discrétisation des horloges [Kwiatkowska et al., 2006] et le second sur les jeux stochastiques [Kwiatkowska et al., 2009]. Les deux techniques souffrent cependant de limitations. Dans le premier cas, les principales restrictions sont l'impossibilité d'utiliser des comparaisons strictes dans les contraintes d'horloges et l'interdiction d'utiliser des comparaisons diagonales (comparaisons entre deux horloges). Ces restrictions ne sont pas en soit bloquantes pour la modélisation des RCsF, mais sont contraignantes lors de la modélisation. En revanche, même si les expérimentations menées dans [Norman et al., 2012] montrent que cette technique a de bonnes performances en pratique (temps d'exécution et occupation mémoire), les auteurs indiquent que les performances se dégradent fortement lorsque la contrainte maximale sur les horloges dans le modèle augmente. C'est un problème dans le cas des protocoles RCsF, car certains des comportements des protocoles sont à courte échéance (acquittement d'un paquet par exemple de l'ordre de la milliseconde) et d'autres à longue échéance (à cause du cycle d'endormissement, de l'ordre de la seconde). La différence d'échelle oblige à avoir de grandes valeurs de contraintes d'horloge pour modéliser les événements longs car les contraintes sont des entiers naturels (les grandes valeurs sont donc des multiples de la plus petite valeur). Dans le cas de la méthode des jeux stochastiques [Kwiatkowska et al., 2009], ce problème n'apparaît pas. En revanche, il est dans ce cas interdit d'utiliser des variables globales. Cela pose problème pour modéliser les protocoles, car la modélisation des échanges de données induites par les communications entre nœuds se fait à l'aide de variables globales.

En plus de ces contraintes induites par les méthodes de vérifications, il existe des restrictions sur le langage de modélisation de PRISM. D'abord, PRISM n'implémente pas de tableaux de variables, c'est un problème pour modéliser la topologie du réseau par exemple. En effet, dans la littérature [Fehnker et al., 2007] [Wibling et al., 2004] [Tschirner et al., 2008] la topologie est représentée par une matrice de connectivité (aussi appelée matrice d'adjacence) : une matrice carrée où chaque ligne et chaque colonne correspondent à un nœud, une valeur c_{ij} de la matrice à 1 indique une connexion entre les nœuds i et j (la valeur à zéro indique qu'il n'y à pas de connexion). Dans PRISM, il faut décrire spécifiquement le voisinage de chaque nœud (il n'est pas possible d'avoir une définition générale d'un nœud puis de l'instancier plusieurs fois), cela oblige à redéfinir chaque nœud puisqu'ils ont un voisinage propre (ou à écrire un outil qui permet de générer le modèle PRISM [Fruth, 2011] mais on verra dans la section 7.4 que cette méthode a aussi des limitations). Il n'est pas non plus possible de réaliser de synchronisation avec passage de valeur, ce qui est utile pour représenter les transmissions de paquets de données. Le traitement des données sur les transitions est limité à des opérations arithmétiques basiques. Pour les opérations plus

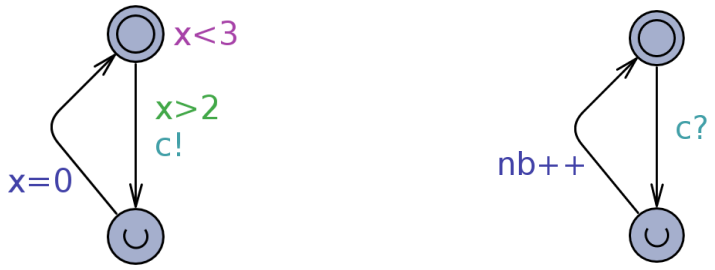
complexes (qui nécessitent des traitements conditionnels sur les données ou bien avec des boucles) il faut générer les résultats à l'avance et encoder les différents cas dans différentes transitions.

Toutes ces restrictions rendent la modélisation et vérification de protocoles temps-réel pour RCsF avec PRISM contraignante. En revanche, PRISM est largement utilisé (principalement pour la vérification de chaînes de Markov et MDP), il existe de nombreux articles sur les algorithmes de vérifications ainsi que sur des cas d'étude. On peut aussi noter qu'une des caractéristiques qui fait la puissance de PRISM (mais qui limite aussi le passage à l'échelle) est sa capacité à quantifier automatiquement la probabilité du pire cas, et pas seulement de vérifier si une probabilité donnée en entrée est respectée.

7.3.3 UPPAAL

L'outil de *Model Checking* UPPAAL [Gerd Behrmann and Larsen, 2004] permet de vérifier des TA. Son extension, UPPAAL-PRO [Engdahl and Haugstad, 2008], permet la spécification et vérification de PTA. UPPAAL offre la possibilité d'implémenter le modèle en langage graphique en plaçant directement les états et les transitions et en leur assignant des invariants, des gardes et des mises à jour (cf. Figure 7.2, dont le fonctionnement est détaillé plus loin dans ce paragraphe). UPPAAL permet de déclarer des variables et tableaux de variables qui sont notamment utiles pour implémenter une représentation de la topologie (matrice de connectivité) du réseau ou bien la file d'attente d'un nœud du réseau par exemple, les données peuvent être locales à un nœud ou bien globales. Il est possible d'écrire des mises à jour sur les données (exécutées lors du passage des transitions) sous forme de fonctions en un langage dérivé du C (il n'y a donc pas besoin d'encoder les différents comportements à l'aide de multiples transitions comme c'est le cas dans PRISM, cf. section 7.3.2). UPPAAL ajoute aussi la notion d'urgence [Gerd Behrmann and Larsen, 2004] aux états : un état urgent est un état dans lequel le temps ne peut pas s'écouler, une transition doit être prise immédiatement (si aucune transition n'est active le système est bloqué). Il est aussi possible de définir des états dits "engagés" (*committed*) qui en plus d'être urgents sont prioritaires, c'est-à-dire que si plusieurs transitions sont actives au même instant, une transition qui sort d'un état "engagé" doit être prise en premier. Les synchronisations entre TA sont réalisées sur des variables appelées canaux : sur une transition, un nœud peut émettre un signal de synchronisation sur une variable canal, un autre nœud dont une transition active possède cette variable canal prend la transition au même instant que le nœud émetteur. Par exemple, sur la Figure 7.2, le nœud émetteur (Figure 7.2(a)) attend entre 2 et 3 secondes (grâce à la garde $x > 2$ et l'invariant $x < 3$) dans l'état initial représenté par un double cercle et émet un signal de synchronisation sur le canal c (noté $c!$), il remet ensuite immédiatement (l'état est urgent, il est marqué avec la lettre "u") son horloge x à zéro en retournant dans son état initial. Le récepteur (Figure 7.2(b)) attend un signal sur le canal c (noté $c?$) sur réception du signal il incrémente la variable `nb` en retournant dans son état initial. Les deux TA passent la transition depuis l'état initial à l'état urgent au même instant (décidé par l'émetteur). UPPAAL permet de modéliser

deux types de synchronisations : les synchronisations un à un comme l'exemple de la Figure 7.2 et les synchronisations *broadcast* qui consistent à synchroniser plusieurs TA sur une transition. Les synchronisations un à un sont bloquantes, c'est-à-dire que l'émission d'un signal doit correspondre à la réception du signal par un autre TA sinon la transition ne peut pas être prise. Ce n'est pas le cas des synchronisations *broadcast* pour lesquels la transition d'émission peut être prise même si aucun TA ne prend la transition qui correspond à la réception du signal. Les synchronisations *broadcast* sont très utiles pour modéliser les transmissions *broadcast* dans les réseaux sans fil comme nous le verrons dans le chapitre suivant.



(a) Emetteur (b) Récepteur

FIGURE 7.2 – Exemple de NTA UPPAAL : émetteur-récepteur

[Engdahl and Haugstad, 2007] et [Engdahl and Haugstad, 2008] présentent un algorithme de vérification des PTA pour UPPAAL. Il est basé sur l'algorithme de [Kwiatkowska et al., 2002a] mais avec une représentation encore plus concise des états qui permet un meilleur passage à l'échelle. A notre connaissance cet algorithme n'impose pas de restrictions sur la modélisation du système. En revanche, il est seulement possible de vérifier des probabilités maximales d'accessibilité (probabilité maximale d'arriver dans un état). On peut noter que dans de nombreux cas on peut exprimer une propriété par rapport à la probabilité maximale ou minimale indifféremment [Berendsen et al., 2010] : la probabilité maximale d'arriver dans un "mauvais" état et probabilité minimale d'être toujours dans un "bon" état sont complémentaires.

Le langage utilisé par UPPAAL est le plus adapté pour modéliser les protocoles temps-réel pour RCsF. De plus UPPAAL est capable de réaliser le *Model Checking* sur les PTA. Nous nous sommes donc tournés vers cet outil pour modéliser et vérifier RTXP. Cependant, l'arrêt de la distribution d'UPPAAL-PRO ainsi que l'absence de documentation de la partie probabiliste sont des points très négatifs sur son utilisation. La partie de vérification temporelle reste quant à elle bien documentée, régulièrement maintenue et dispose d'une large communauté d'utilisateur.

7.3.4 Conclusion sur les outils

Au niveau des outils, UPPAAL semble le plus à même de nous permettre de modéliser correctement les RCsF. Cependant il existe un vrai manque au niveau de la vérification des propriétés probabilistes (absence de documentation). PRISM en revanche a des lacunes à cause des restrictions dues aux techniques de vérification et

au langage de modélisation. Nous concluons qu'aucun des outils n'est complètement satisfaisant pour la vérification des PTA. En revanche, UPPAAL permet de vérifier des propriétés temporelles sur des TA donc des modèles de RCsF sans la prise en compte du lien radio non-fiable. Dans le chapitre suivant, pour palier à ce manque, nous proposons une technique qui permet de modéliser le lien radio non-fiable et qui est indépendante de l'outil de vérification utilisé.

7.4 Vérification des RCsF

Dans cette section, nous nous intéressons à la littérature sur la vérification formelle de protocoles pour RCsF. D'abord, on constate que la littérature n'est pas très fournie dans ce domaine. Ensuite, on remarque qu'il existe peu d'études de vérification qui incluent la modélisation des comportements probabilistes des protocoles et/ou du canal radio. Enfin la plupart des études portent sur des réseaux de quelques nœuds (pas de passage à l'échelle).

7.4.1 Vérifications non probabilistes

Nous commençons par décrire les études qui ne modélisent pas l'aspect probabiliste du lien radio (probabilité qu'un paquet ne soit pas reçu) et qui utilisent en majorité l'outil UPPAAL et les NTA pour modéliser les RCsF. Dans ces travaux, chaque nœud est modélisé par un TA et les transmissions sont modélisées par des synchronisations entre TA.

Dans [Watteyne et al., 2006a], les auteurs proposent et vérifient formellement le protocole MAC Dual-Mode présenté dans la section 2.2.1 du Chapitre 2. Les transmissions sont modélisées par des synchronisations avec un TA dédié, il attend des synchronisations des nœuds, quand il s'en produit une, le TA synchronise seulement les récepteurs en fonction de la topologie du réseau (représentée sous forme de matrice de connectivité). Cela permet de vérifier que le comportement temporel de Dual-Mode est correct.

Dans [Tschirner et al., 2008] les auteurs étudient la QoS dans les réseaux de capteurs biomédicaux. Ils utilisent le *Model Checker* UPPAAL non seulement pour vérifier formellement des propriétés de QoS mais aussi pour simuler le modèle étudié. Les résultats de vérification et simulation d'UPPAAL sont comparés avec ceux obtenus grâce au simulateur à événement discret OMNeT++ [Varga et al., 2001]. Les auteurs choisissent de modéliser une version simplifiée la machine à état de la radio Chipcon CC2420 [TI, 2007]. La couche MAC est une modélisation du CSMA/CA : un nœud s'il veut émettre, le fait directement si le canal est libre. Sinon il attend en échantillonnant le canal régulièrement, puis envoie son paquet (rien empêche que deux nœuds commencent à envoyer au même instant et aucune résolution de collision n'est implémentée). Le protocole de routage consiste en une inondation du réseau. Les auteurs évaluent la connectivité du réseau et le taux de livraison. Des résultats de vérification sont donnés pour des réseaux de 5, 6 et 11 nœuds qui ont des périodes fixes d'émission de trafic de données. Il est à noter que, pour la vérification, le lien

radio non-fiable n'est pas modélisé (un paquet est toujours reçu si aucun autre paquet n'est transmis par les voisins). Les transmissions sont représentées par des synchronisations, il y a collision si un nœud est synchronisé de nouveau avant la fin d'un paquet. Les résultats de vérification sont commentés succinctement, les auteurs ne précisent malheureusement pas s'il détectent des erreurs dans la conception de leur protocole ou bien si la non vérification des propriétés vient de la topologie utilisée. Les auteurs comparent ensuite les résultats de simulations (taux de livraison) réalisées avec UPPAAL et OMNeT++ sur deux topologies de 8 nœuds. Le taux de livraison est, sans surprise, largement plus élevé dans le cas d'UPPAAL car la couche physique n'est pas représentée fidèlement. Malheureusement, les résultats de simulations et de vérifications ne sont pas comparés. On note cependant que les taux de livraisons atteignables lors des vérifications sont très optimistes par rapport à ceux des simulations réalisés avec OMNeT++, c'est encore dû au manque de réalisme du modèle UPPAAL. Ces résultats sont tout de même un premier pas vers la vérification formelle de protocoles de RCsF, les résultats montrent l'importance de la qualité de modélisation, qui d'une part contribue à l'explosion combinatoire (plus on met de détails dans le modèle plus le TLTS induit est grand) mais d'autre part contribue à obtenir des résultats de vérification réalistes. On décèle aussi dans cet article un problème de passage à l'échelle (vérification sur des réseaux de 6 nœuds et cas d'étude de simulations avec 8 nœuds), cela reste cependant réaliste pour des réseaux de capteurs biomédicaux [IEEE, 2012].

Les auteurs de [Fehnker et al., 2007] vérifient, à l'aide du *model checker* UPPAAL, LMAC [van Hoesel and Havinga, 2004], un protocole MAC pour RCsF. Dans LMAC, le temps est découpé en *slots* qui sont regroupés en trames. Le nombre de *slots* par trame est déterminé en fonction de la topologie du réseau (il doit être plus grand que le plus grand domaine d'interférence pour laisser la possibilité d'allouer un *slot* à chaque nœud), les *slots* sont attribués de manière distribuée. Le principe de LMAC est très simple, chaque nœud essaye de s'attribuer un *slot* de la trame, s'il détecte qu'il est en collision avec un autre nœud, il recommence (le protocole n'est pas décrit en détail ici car le but de cette section est de décrire le processus de vérification, pour plus de détails sur LMAC, voir [van Hoesel and Havinga, 2004]). Chaque nœud est modélisé par un TA d'UPPAAL. La topologie est représentée par une matrice de connectivité, les transmissions de paquets sont modélisées grâce à des synchronisations, si un nœud est synchronisé une seconde fois durant un *slot* il y a collision. Les auteurs vérifient le protocole sur toutes les topologies connectées de 4 et 5 nœuds. Les propriétés vérifiées sont : l'absence de blocage, la synchronisation des voisins (ils doivent être d'accord sur le numéro du *slot* courant), l'accessibilité d'une configuration (on rappelle qu'une configuration est un état du TLTS dans le cas des NTA, cf. section 7.2.6) où tous les 2-voisins ont choisi un *slot* différent, le fait que tous nœuds qui sont en collision finissent forcément par choisir un *slot* différent. Les résultats du *Model Checking* indiquent que toutes les propriétés, excepté la dernière, sont vérifiées sur toutes les topologies. En effet, des topologies pour lesquelles les collisions ne peuvent pas être signalées aux nœuds incriminés sont mises en évidence (la collision n'est perçue par aucun nœud : un des nœuds incriminés est en collision avec son seul voisin donc déconnecté du réseau). Les auteurs en changeant les règles d'affectation des *slots* permettent de résoudre ce problème. La vérification formelle de LMAC permet donc de mettre en évidence

des problèmes d'allocations de *slots* pour certaines topologies et grâce aux traces d'erreur, de résoudre ces problèmes. C'est un exemple très intéressant d'application de méthodes formelles aux RCsF car il permet d'en montrer l'utilité (mise en évidence d'erreurs subtiles) tout en en décrivant les limites (passage à l'échelle). L'idée qui nous semble ici la plus intéressante est celle de vérifier de nombreuses topologies (toutes les topologies connectées), car dans les RCsF les liens non-fiables font évoluer la topologie, vérifier toutes les topologies connectées permet donc de se prémunir contre tous les cas de figure.

[Mounier et al., 2007] est également une étude sans la modélisation de l'aspect probabiliste du lien radio. Contrairement aux études précédemment présentées, ce travail utilise le langage formel IF [Bozga et al., 2004] pour représenter le RCsF. IF est un langage de description de spécifications, il se place comme un langage intermédiaire entre les standards de haut-niveau (UML [Rumbaugh et al., 1999], SDL [Ellsberger et al., 1997], etc) et les outils de vérification [Bozga et al., 2004]. Le modèle du système est composé de plusieurs processus qui s'exécutent en parallèle et interagissent grâce à des variables partagées et/ou des files de messages, chaque processus est un TA (décrit dans la section 7.2.6). Dans [Mounier et al., 2007] les auteurs proposent de trouver la durée de vie minimum d'un RCsF par *Model Checking*. Plusieurs définitions de la durée de vie du réseau sont étudiées : mort (batterie vide) du premier nœud, réseau déconnecté, un pourcentage arbitraire du réseau est mort. Deux protocoles de routage sont modélisés tour à tour : une inondation et le protocole Directed Diffusion [Intanagonwiwat et al., 2000]. Les nœuds sont modélisés par des processus IF qui sont équipés d'une file de messages FIFO, les communications entre nœuds sont des passages de messages entre les processus. Un processus qui contient la topologie sous forme de matrice de connectivité permet de diriger le paquet courant vers les files des nœuds qui sont effectivement connectés (à portée radio) au nœud émetteur. Le détail de modélisation des collisions n'est pas donné. Pour la vérification, les auteurs, après des essais sur des réseaux de 4 à 11 nœuds, choisissent d'utiliser un réseau de 6 nœuds car le temps de vérification reste raisonnable (6 heures contre 2 jours pour les réseaux de 11 nœuds). Les résultats pour les différentes définitions de la durée de vie montrent que le routage par inondation à une durée de vie minimum inférieure à celle du routage avec le protocole Directed Diffusion. Comme pour les autres études, la taille des réseaux vérifiés reste modeste.

7.4.2 Vérification probabiliste des RCsF

Nous nous intéressons ensuite aux études qui prennent en compte les comportements probabilistes des protocoles pour la modélisation et la vérification des RCsF.

Les auteurs de [Olveczky and Thorvaldsen, 2006] utilisent le langage RT-Maude pour modéliser, puis vérifier, le protocole OGDC qui permet de réaliser du contrôle de topologie dans les RCsF. Le but de OGDC est de sélectionner des nœuds actifs, de manière distribuée, de telle manière que la superposition de couvertures (la couverture étant l'aire couverte par le capteur physique) soit la plus petite possible. Les auteurs modélisent les nœuds qui exécutent OGCD grâce à RT-Maude, un langage basé sur la logique de réécriture [Meseguer, 1992]. La logique de réécriture a pour but

d'unifier les différents modèles de calcul et de concurrence existants [Meseguer, 1992] (en fait, tous les formalismes décrits dans la section 7.2 peuvent être exprimés en RT-Maude [Meseguer, 1992]). Les deux outils de modélisation sont, les équations qui permettent de décrire des opérations déterministes, et les règles de réécriture qui permettent de décrire du non-déterminisme (concurrence dans les réseaux). Dans [Olveczky and Thorvaldsen, 2006], les actions internes aux nœuds sont donc représentées par des équations et les actions de communication sont représentées par des règles de réécriture. Les nœuds sont modélisés sous forme d'objets sur lesquels s'appliquent les équations et les règles de réécriture. Ils sont identifiés par leur position géographique, la connectivité est modélisée en vérifiant que les récepteurs d'un message sont à portée de l'émetteur dans la règle de réécriture qui correspond à la transmission d'un paquet. Les collisions sont modélisées de la même manière : en vérifiant dans la règle de transmission qu'il n'y a pas de réception en cours. Les comportements probabilistes (choix du premier nœud actif, par exemple) sont représentés en échantillonnant l'espace des possibles, c'est-à-dire que seulement certains comportements (choisis à l'aide d'un générateur pseudo-aléatoire) sont envisagés. La vérification n'est donc pas exhaustive, il existe une probabilité pour que le système ait un comportement non pris en compte durant la vérification. Les propriétés vérifiées sont : le temps minimum et maximum de convergence de OGCD (moment où toute la zone est couverte) et la couverture de toute la zone d'intérêt. Ces propriétés sont vraies pour les réseaux évalués. Cependant, les topologies considérées sont seulement de 6 et 5 nœuds.

L'auteur de [Fruth, 2006] utilise le *Model Checker* probabiliste PRISM pour vérifier un modèle de la norme 802.15.4 [IEEE, 2011b] présenté dans la section 2.1.1 du Chapitre 2. Le protocole est modélisé à l'aide d'un PTA, le réseau constitue donc un NPTA. La topologie consiste en deux émetteurs et deux récepteurs qui sont tous à portée radio, elle est encodée avec un PTA dédié qui attend des synchronisations des émetteurs et synchronise les récepteurs. On peut noter que l'aspect probabiliste du lien radio n'est pas modélisé. En revanche, le comportement probabiliste du protocole est lui modélisé : notamment la valeur du temps de *backoff* est choisie aléatoirement. Dans les scénarios étudiés, chaque station émettrice a un seul paquet à émettre, l'auteur vérifie : la probabilité minimum que les deux stations finissent par envoyer leur paquet, la probabilité maximum d'avoir au moins k collisions, l'espérance maximum du nombre de collisions avant que les deux stations réussissent à envoyer leurs paquets et l'espérance maximum du temps avant que les deux nœuds envoient leur paquet. Les résultats montrent que dans certains cas, notamment pour des trames courtes et des paquets longs l'espérance maximum du temps pour transmettre un paquet est infinie (ce qui signifie qu'en moyenne la collision n'est pas résolue). De manière générale les résultats présentés constituent des bornes prouvables sur le fonctionnement de 802.15.4. Cependant, ces résultats ne sont valables que pour des réseaux de deux émetteurs qui transmettent seulement un paquet, ce qui restreint leur portée.

Dans [Fruth, 2011], l'auteur décrit l'outil CaVi, qui permet de modéliser graphiquement la topologie d'un réseau sans fil puis de la traduire en modèle PRISM. CaVi permet à l'utilisateur de définir la position des nœuds, l'outil calcule ensuite la qualité des liens en fonction du placement des nœuds. La qualité des liens dépend de la propagation et des interférences des autres nœuds. A partir de cette spécification gra-

phique, un modèle PRISM est généré, la topologie du réseau est une clique (les nœuds sont tous à portée les uns des autres) et les probabilités de succès des transmissions entre chaque paire de nœuds sont exprimées. Dans l'état actuel, l'outil ne permet pas de spécifier le comportement des protocoles à vérifier, le modèle généré inclut obligatoirement un routage par inondation et ne modélise pas de mécanisme MAC (les nœuds accèdent de manière concurrente au médium, possiblement tous à la fois). Des tests sont réalisés dans [Fruth, 2011] avec des réseaux de 10 nœuds mais seulement pour cette pile protocolaire très simple. Pour des réseaux de plus de 20 nœuds, PRISM n'est pas capable de lire le fichier généré car il est trop grand (le nombre de paires de nœuds augmente de manière exponentielle avec le nombre de nœuds : on peut noter que l'utilisation de notations plus concises sous forme de tableaux permettrait d'atténuer ce problème, mais PRISM ne permet pas leurs utilisation). Cet outil mériterait d'être testé avec d'autres piles protocolaires, il faudrait cependant pouvoir le modifier pour cela et à notre connaissance il n'est pas distribué.

Les études formelles de protocoles pour RCsF présentées dans cette section soulignent deux problèmes principaux : le manque de réalisme de la modélisation du lien radio et la difficulté du passage à l'échelle. Ces deux aspects sont détaillés dans le Chapitre 8 et des solutions sont proposées.

7.5 Conclusion

Dans ce chapitre nous avons fait un tour d'horizon des différentes méthodes, formalismes et outils disponibles pour la vérification formelle et nous nous sommes concentrés sur ceux qui permettent de vérifier des protocoles temps-réel pour les RCsF. Le *Model Checking* nous semble la méthode la plus appropriée car elle permet une vérification exhaustive et automatique des comportements possibles du système et fournit des contre-exemples en cas de non respect des spécifications. Elle souffre cependant du problème d'explosion combinatoire ce qui est problématique pour la vérification des RCsF (du fait du grand nombre de nœuds qui composent ce type de réseaux) comme nous le verrons dans les chapitres suivants. Il existe de nombreux formalismes pour lesquels le *Model Checking* est décidable, mais ce sont les PTA qui semblent les plus adaptés pour modéliser tous les aspects des RCsF. En effet, les PTA permettent de représenter le temps, les probabilités, la concurrence, les communications et le traitement des données. En revanche, les outils existants pour la vérification des PTA manquent de maturité. Fortuna est, à notre connaissance, très peu utilisé et ne possède pas de documentation. PRISM au contraire est bien documenté et largement utilisé, mais plus pour la vérification de chaînes de Markov et MDP, de plus les limitations imposées par les techniques de vérifications et par le langage de modélisation restreignent son usage. UPPAAL est un outil pour le *Model checking* temporel de spécifications exprimées en TA, son langage de modélisation permet d'exprimer efficacement tous les aspects des RCsF. De plus, il existe une extension qui permet de réaliser la vérification des PTA, malheureusement, cette extension n'est pas documentée et n'est plus distribuée. Dans la suite de ce document, nous utilisons tout de même UPPAAL pour modéliser et vérifier les RCsF du fait de la puissance de son

langage de modélisation. Cela nous permet de vérifier le comportement temporel des protocoles étudiés de manière satisfaisante. Nous développons ensuite une méthode pour ajouter la prise en compte des liens radios quel que soit l'outil de vérification utilisé, cette méthode est basée sur la génération de "topologies possibles" (dues à l'apparition et à la disparition de liens radios), elle sera détaillée dans le Chapitre 8.

On recense peu de vérifications formelles de protocoles pour RCsF dans la littérature. Les études proposées ne prennent pas, pour la plupart, en compte l'aspect probabiliste du lien radio et sont limitées à la vérification de réseaux de quelques nœuds seulement. De plus les styles de modélisation dépendent beaucoup des protocoles modélisés. Notre but est d'avoir une approche plus globale et d'émettre des recommandations sur la modélisation des RCsF. C'est pourquoi nous proposons une étude sur la représentation des transmissions *broadcast* dans le Chapitre 8.

Dans le chapitre suivant, nous proposons une méthodologie en trois points qui permet de répondre à trois problématiques de la vérification formelle des RCsF : la modélisation de la nature *broadcast* médium radio, la vérification prenant en compte l'aspect probabiliste du lien radio et le passage à l'échelle. Nous appliquons ensuite cette méthodologie à RTXP dans le Chapitre 9.

Chapitre 8

Méthodologie de vérification formelle de protocoles de communications pour réseaux de capteurs sans fil

Dans ce chapitre, nous proposons une méthodologie de vérification formelle des protocoles de commnuications temps-réel pour RCsF. Nous présentons d'abord l'impact sur la vérification formelle des caractéristiques des RCsF :

- le lien radio sans fil de nature *broadcast* ;
- le lien radio également non-fiable ;
- la taille des RCsF, qui peuvent être composés de plusieurs centaines de nœuds.

Comme on l'a constaté dans le chapitre précédent ces points sont peu abordés dans la littérature. Ensuite, nous décrivons trois contributions qui permettent de répondre à ces aspects :

- Une étude des performances du *Model Checking* avec différentes modélisations des transmissions *broadcast* utilisées dans la littérature. Cette étude conclut qu'une des méthodes limite l'explosion combinatoire.
- Une méthode de vérification qui permet de prendre en compte des liens radios réalistes. Cette méthode est indépendante de l'outil de vérification utilisé, elle reste néanmoins seulement applicable à de petits réseaux (une dizaine de nœuds).
- Une méthode qui combine *Network Calculus* et *Model Checking* et qui permet de passer l'échelle tout en gardant un niveau de confiance acceptable. En effet la méthode utilise le *Model Checking* pour vérifier que les hypothèses de service (c'est-à-dire la capacité d'un nœud à servir le trafic qu'il reçoit) de *Network Calculus* sont atteignables par le modèle du protocole étudié.

Ces contributions sont ensuite articulées pour former une méthodologie de vérification formelle de protocoles temps-réel pour les RCsF. Cette méthodologie est formée d'une étape de modélisation qui tire parti de l'étude des modélisation du *broadcast* et de trois étapes de vérification :

1. sans les liens non-fiables et sur des réseaux de quelques nœuds seulement ;
2. avec les liens non-fiables toujours sur des réseaux de quelques nœuds ;
3. sur de grand réseaux (plusieurs centaines de nœuds).

Ces trois étapes de vérification permettent d'augmenter considérablement le niveau de sûreté des applications critiques des RCsF.

8.1 Spécificités des réseaux de capteurs sans fil

Dans la littérature scientifique, on trouve de nombreux cas de vérifications formelles de protocoles [Clarke and Wing, 1996]. Cependant, comme évoqué dans le chapitre précédent, très peu d'études portent sur des protocoles sans fil et encore moins sur des réseaux larges échelles. Dans cette section nous décrivons les aspects spécifiques aux RCsF qui posent problème lors de la vérification des protocoles temps-réel pour RCsF. Nous en comptons trois : la nature *broadcast* des communications sans fil (transmissions *broadcast*), le lien radio non-fiable et la grande taille des RCsF.

8.1.1 Nature *broadcast* du médium radio

Dans les réseaux sans fil, lorsque un nœud émet un message, il est reçu par tous les nœuds à portée radio. C'est un aspect important des réseaux sans fil, car les communications et les interférences dépendent de la topologie logique du réseau (les voisinages). Il faut donc être capable d'une part de modéliser la topologie et d'autre part, de restreindre les communications aux nœuds d'un même voisinage, et les interférences seulement aux nœuds interférants. Dans la littérature, on trouve plusieurs façons de modéliser ces interactions locales. Dans la section 8.2 nous proposons d'étudier ces modélisations. Nous nous intéressons à la taille de l'espace de configurations à explorer que chacune de ces modélisations induit. Cet aspect est déterminant pour les performances du *Model Checking*. En effet, plus l'espace de configurations est grand, plus son exploration exhaustive est longue. De plus si la taille de l'espace de configurations est trop importante, elle ne tient pas dans la mémoire de la machine, la vérification est alors impossible. L'étude réalisée montre que l'une des méthodes de modélisation engendre un espace de configurations plus petit que les autres méthodes. Cette méthode devra donc être préférée pour la modélisation des RCsF pour limiter l'explosion combinatoire de l'espace de configurations et donc pouvoir vérifier des réseaux plus grands.

8.1.2 Liens radio non-fiable

Comme on l'a vu dans la section 3.2.1 du Chapitre 3, le lien radio n'est pas fiable. Des pertes de paquets peuvent apparaître en raison des interférences entre nœuds et

des atténuations aléatoires dues aux réflexions (*multipath-fading*) et diffractions (*shadowing*) sur des éléments de l'environnement. Pour les interférences, nous choisissons comme dans les chapitres 4, 5 et 6 d'utiliser un modèle d'interférences à deux sauts. En ce qui concerne les atténuations aléatoires du signal, nous utilisons le modèle *log-normal shadowing* présenté dans la section 3.2.1. Il modélise les atténuations aléatoires de la puissance reçue grâce à une loi log-normale. Le phénomène d'atténuation aléatoire implique qu'il est possible de perdre un paquet avec une probabilité connue. Le comportement du modèle du RCsF dépend donc de cette probabilité : si le paquet est perdu, les nœuds agissent d'une certaine façon (retransmissions) et si le paquet n'est pas perdu, ils ont un autre comportement (absence de retransmissions). Comme on l'a vu dans le chapitre précédent, des formalismes existent pour représenter les comportements qui dépendent de lois de probabilités. Cependant, les outils associés ne permettent pas de représenter les RCsF de manière satisfaisante (limitations liées aux techniques de vérification et au langage de modélisation). Dans la section 8.3, nous proposons une méthode qui permet de prendre en compte les liens non-fiables et qui est indépendante de la méthode de vérification. La fluctuation des liens modifie continuellement la topologie du réseau, nous proposons donc de générer toutes les topologies possibles en leur associant une probabilité d'existence qui est calculée à partir du modèle de propagation radio (nous utilisons *log-normal shadowing*). Pour chaque topologie, le fonctionnement du protocole est vérifié, cela permet de connaître avec quelle probabilité la propriété vérifiée est vraie.

8.1.3 Taille des RCsF

Le troisième point problématique pour la vérification de protocoles temps-réel pour les RCsF est leur taille qui peut atteindre plusieurs centaines voir milliers de nœuds [Augé-Blum et al., 2012], avec la possibilité d'avoir des voisinages de plusieurs dizaines de nœuds. Les protocoles développés pour ce type de réseaux doivent donc être capables de passer l'échelle, la vérification formelle des protocoles implique par conséquent de modéliser et vérifier de grands réseaux. Comme évoqué dans le Chapitre précédent, le principal inconvénient du *Model Checking* est l'explosion combinatoire de l'espace de configurations qui empêche le passage à l'échelle de la méthode. Une "bonne" modélisation, comme pour le cas du *broadcast* présenté dans la section 8.2, permet de limiter cette explosion combinatoire, elle reste néanmoins présente. Dans la section 8.4, nous présentons une méthode de vérification qui allie *Network Calculus* et *Model Checking* pour garder un niveau de confiance élevé tout en augmentant le passage à l'échelle.

8.2 Etude de la modélisation des transmissions *broadcast*

Dans cette section, nous nous intéressons à la modélisation de la nature *broadcast* des transmissions sans fil. Dans un réseau sans fil, quand un nœud transmet un paquet, ce paquet est reçu par tous les nœuds à portée radio, ce type de transmission est

nommé *broadcast*. Dans la littérature sur la vérification formelle des réseaux sans fil, on trouve trois façons de modéliser ce type de transmissions, nous les présentons dans la section 8.2.1. Le fait qu'il existe plusieurs façons de modéliser le même comportement nous pousse à nous demander si une représentation est meilleure que les autres pour les performances de la vérification (pour le passage à l'échelle notamment). Dans cette section nous évaluons l'impact du choix de modélisation du *broadcast* sur les performances du *Model Checking*. La meilleure technique étant celle qui permet de modéliser correctement cet aspect tout en limitant le plus possible l'explosion combinatoire de l'espace de configurations, autorisant ainsi à vérifier des réseaux plus grands.

La méthodologie d'évaluation se déroule en deux étapes :

1. Dans un premier temps, nous prenons un modèle minimaliste * de chaque modélisation de *broadcast*. Nous décrivons ensuite l'espace de configurations (une configuration correspond à un état du TLTS induit par le TA comme mentionné dans la section 7.2.6 du chapitre 7) générées lors du *Model Checking*. Cela nous permet de déterminer quelle modélisation induit le plus petit espace de configurations (donc le plus rapide à explorer).

2. Dans deuxième temps, nous produisons 3 modèles du protocole de routage GRAB [Ye et al., 2005a] présenté dans la section 2.1.2 du Chapitre 2, un pour chaque façon de représenter les transmissions *broadcast*. Nous vérifions ensuite le comportement temporel du protocole et évaluons les performances d'UPPAAL en termes de taille d'espace de configurations généré et de durée de vérification. Cela nous permet de conforter les conclusions obtenues avec les modèles minimalistes.

Nous décrivons d'abord les trois types de modélisation de *broadcast* en nous appuyant sur leur utilisation dans la littérature.

8.2.1 Modélisations du *broadcast*

On trouve dans la littérature trois façons de représenter le *broadcast*, toutes sont basées sur des synchronisations entre les différents nœuds du réseau. Dans [Gerd Behrmann and Larsen, 2004], un tutoriel sur l'utilisation de UPPAAL, les auteurs préconisent de représenter les communications entre nœuds d'un réseau grâce a des synchronisations entre TA avec passage de valeur : cela consiste à synchroniser des TA comme décrit dans la section 7.3.3 du Chapitre 7 en modifiant des variables globales qui représentent le contenu du message. Les modélisations des transmissions *broadcast* dans les réseaux sans fil que nous présentons dans cette section s'inspirent de ce principe. Dans les réseaux radio, la portée du paquet doit cependant être limitée aux voisins du nœud, la topologie logique entre donc en considération dans la modélisation des transmissions *broadcast*.

∗. Nous définissons un modèle minimaliste comme étant une représentation de la transmission seule, sans aucun autre aspect protocolaire.

Vérification de la connectivité après synchronisation

Dans [Wibling et al., 2004] les auteurs modélisent un réseau adhoc avec UPPAAL pour évaluer la faisabilité de la vérification par *Model Checking* sur ce type de réseaux. Ils modélisent la topologie du réseau grâce à une matrice de connectivité (dans UPPAAL c'est un tableau de booléens qui fait partie des variables globales). Le médium du réseau est représenté par un canal de synchronisation *broadcast*, le nœud émetteur synchronise tous les nœuds du réseau, chaque nœud vérifie ensuite s'il est connecté à l'émetteur grâce à la matrice de connectivité. S'il est connecté, il copie la valeur de la variable globale contenant la donnée du message dans une variable locale, sinon il ne prend pas en compte la variable globale. Cette méthode est aussi utilisée dans [Tschirner et al., 2008], détaillé dans la section 7.4 du chapitre précédent. On peut noter que cette méthode permet de représenter les changements topologiques par des modifications de la matrice de connectivité.

Un canal de synchronisation par groupe de nœuds formant une clique

Les auteurs de [Godskesen and Gryn, 2006] proposent de modéliser les transmissions *broadcast* grâce à l'utilisation de multiples canaux de synchronisation définis en fonction de la topologie. Chaque nœud appartient à un ou plusieurs groupes, les groupes sont des cliques maximales (une clique maximale est une clique que l'on ne peut pas étendre en ajoutant des nœuds adjacents à ceux qui font partie de la clique). Un canal de synchronisation *broadcast* est défini pour chaque groupe, un nœud voulant transmettre synchronise les groupes auxquels il appartient. Un nœud récepteur se synchronise sur les canaux qui correspondent aux groupes auxquels il appartient. Le détail de la construction des groupes n'est pas présenté dans l'article. Une version simplifiée de cette modélisation du *broadcast* consiste à définir les groupes non plus comme des cliques maximales mais comme des paires de nœuds connectés. Dans ce cas, il faut un canal de synchronisation pour chaque arête du graphe qui représente la topologie du réseau. C'est l'approche adoptée par [Watteyne et al., 2006a] et [Fruth, 2006], présentés dans la section 7.4 du chapitre précédent.

Vérification de la connectivité avant synchronisation

Dans [Fehnker et al., 2007], comme dans [Wibling et al., 2004] décrit dans la section 7.4, le médium de communication est représenté par un canal de synchronisation *broadcast* et la topologie par une matrice de connectivité. En revanche, dans [Fehnker et al., 2007], la synchronisation des nœuds récepteurs est effectuée seulement si les nœuds sont voisins dans la topologie (on rappelle que dans [Wibling et al., 2004] tous les nœuds du réseau sont synchronisés) : cette synchronisation conditionnelle est possible en vérifiant la connectivité dans la garde de la transition de synchronisation (qui est donc active seulement si le nœud est connecté à l'émetteur).

8.2.2 Evaluation de la taille des espaces de configurations induits par les modèles de *broadcast* : cas minimalistes

Dans cette section, nous étudions la taille de l'espace de configurations induit par chacune des trois modélisations possibles des transmissions *broadcast*. Comme nous l'avons vu dans la section 7.2.1 du Chapitre 7, lors du *Model Checking*, c'est l'interprétation sémantique du modèle du système qui est explorée (c'est-à-dire l'espace des configurations). Il faut donc être vigilent lors de la modélisation du système pour ne pas produire un modèle qui induit une explosion combinatoire de l'espace des configurations. Nous nous intéressons donc à la taille de l'espace de configurations produit par les 3 façons de modéliser les transmissions *broadcast*. Pour cela nous définissons 3 modèles de *broadcast* minimalistes : ils ne représentent que les transmissions (sans aspects protocolaires). Ces modèles minimalistes permettent de se concentrer seulement sur les configurations induites par les transmissions *broadcast*.

Pour pouvoir constater que les transmissions de paquets dépendent de la topologie (les nœuds à portée radio peuvent communiquer), nous utilisons une topologie simple représentée par la Figure 8.1 : le nœud s a un paquet à émettre et les autres nœuds attendent des paquets (il sera reçu par ses voisins $n1$, $n2$ et $n3$ mais pas par $n4$).

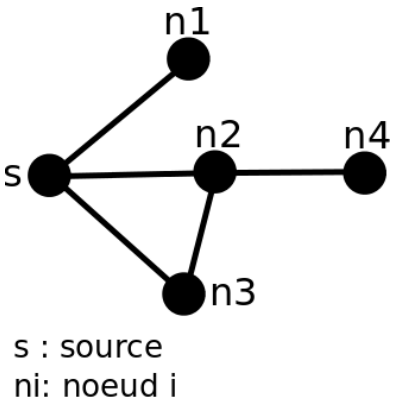
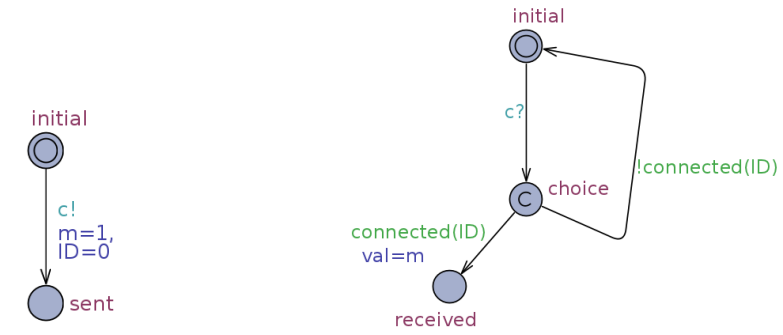


FIGURE 8.1 – Topologie utilisée par les exemples minimalistes

Modèles minimalistes de transmissions *broadcast*

Pour les trois modèles minimalistes, les nœuds sont représentés à l'aide de TA UPPAAL, le canal est modélisé par une variable canal *broadcast* et les paquets sont composés de l'identifiant du nœud émetteur et d'une donnée (ces deux informations sont des entiers) qui sont échangés à l'aide de variables globales (comme préconisé dans [Gerd Behrmann and Larsen, 2004]).

1. Vérification de la connectivité après synchronisation : l'émetteur synchronise tous les nœuds du réseau, puis les nœuds vérifient s'ils sont connectés à l'émetteur et s'ils doivent, par conséquent, récupérer les valeurs des variables représentant le paquet.

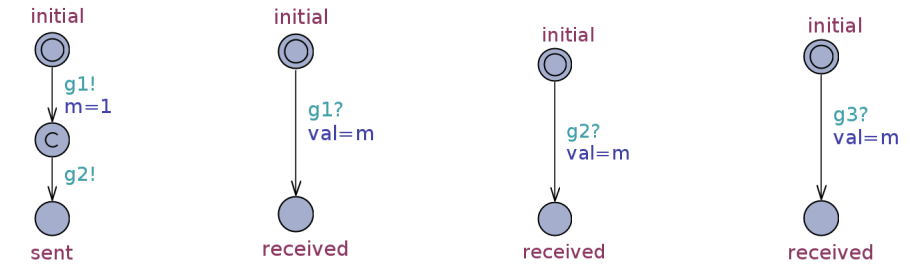


(a) source (b) nœud

FIGURE 8.2 – Modélisation UPPAAL du cas où la connectivité est vérifiée après synchronisation

Les Figures 8.2(a) et 8.2(b) représentent respectivement les TA de la source et des récepteurs (les TA de tous les récepteurs sont similaires). Lors du passage de son unique transition, la source émet un signal de synchronisation (`c!`) et inscrit son identifiant dans la variable globale ID et une donnée (ici l'entier 1) dans la variable globale m. Les nœuds reçoivent simultanément le signal de synchronisation et arrivent dans l'état *choice*. Cet état est un état *committed*, le temps ne peut donc pas s'y écouler, une transition valide doit être prise immédiatement. La condition de validité de la transition est la valeur de *connected(ID)*, une fonction qui renvoie VRAI si le nœud est connecté avec la source ID et FAUX sinon : si la valeur est VRAI, le nœud prend la transition vers l'état *received* et stocke la valeur de m dans sa variable locale *var* sinon, il retourne à l'état *initial* puisqu'il n'est pas à portée de l'émetteur.

2. Synchronisation par groupe de nœuds formant une clique : dans ce cas une variable canal *broadcast* par groupe est définie. Le TA de chaque nœud doit donc être produit en fonction de la topologie du réseau, car ses variables de synchronisation dépendent des groupes auxquels il appartient. Les groupes sont des cliques maximales, sur la topologie de la Figure 8.1 il y en a trois : $g1 = (s, n2, n3)$, $g2 = (s, n1)$ et $g3 = (n2, n4)$.

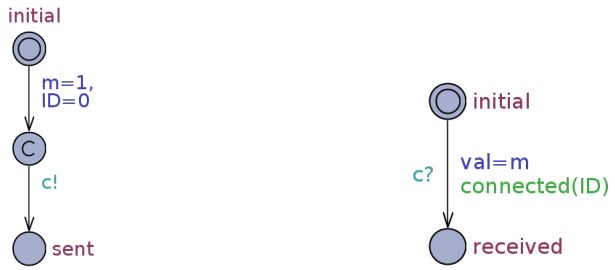


(a) source (b) nœud du groupe 1 (c) nœud du groupe 2 (d) nœud du groupe 3

FIGURE 8.3 – Modélisation UPPAAL du cas des groupes

Les TA de la source, des nœuds du groupe $g1$, $g2$ et $g3$ sont représentés respectivement par les Figures 8.3(a), 8.3(b), 8.3(c) et 8.3(d). A chaque groupe correspond une variable canal *broadcast* (notés `g1`, `g2` et `g3`). La source émet un signal de synchronisation seulement sur les variables canal qui correspondent aux groupes auxquels elle appartient ($g1$ et $g2$), la variable m est mise à jour une seule fois car elle doit avoir la même valeur pour tous les groupes (une seule donnée est envoyée). La variable globale ID n'a pas besoin d'être mise à jour, car dans cet exemple l'information sur la topologie est directement encodée dans les transitions des TA.

3. Vérification de la connectivité avant synchronisation : cette solution est très similaire à la première, mais les mises à jour de l'émetteur sont faites avant la synchronisation et une garde empêche les nœuds non connectés à l'émetteur de se synchroniser avec lui.



(a) source (b) nœud

FIGURE 8.4 – Modélisation UPPAAL du cas où la connectivité est vérifiée avant synchronisation

Le TA de la source et des nœuds sont illustrés respectivement par les Figures 8.4(a) et 8.4(b). La source lors de sa première transition met à jour les variables ID et m. Elle prend ensuite la deuxième transition sans laisser s'écouler de temps (la transition est *committed*). Durant cette transmission, le signal de synchronisation sur la variable canal *broadcast* est émis. Les nœuds récepteurs passent la transition de synchronisation seulement si elle est valide, c'est-à-dire si *connected(ID)* retourne VRAI.

Nombre de configurations accessibles

Pour déterminer laquelle des 3 solutions de modélisation des transmissions *broadcast* atténue le plus le problème de l'explosion combinatoire de l'espace des configurations, nous évaluons la taille en nombre de configurations des espaces générés par les NTA de chaque solution. Comme décrit dans la section 7.2.6, une configuration correspond à un tuple $(\bar{l}, v)$ où $\bar{l}$ est un vecteur d'état de TA et v des valeurs d'horloges des TA. Nous ne modélisons pas de conditions temporelles dans les exemples minimalistes, nous nous intéressons donc seulement au terme $\bar{l}$.

Avec la solution 1 (Figure 8.2), le nombre de configurations accessibles est 18 : la configuration initiale est $\bar{l}$ =*(initial, initial, initial, initial, initial)*, après synchronisation $\bar{l}$ devient *(sent, choice, choice, choice, choice)* car tous les automates passent

la transition de manière simultanée. Ensuite, il y a un entrelacement des transitions pour passer de l'état *choice* à *received* ou *initial* : cela produit $2^4 = 16$ configurations. Cet entrelacement implique donc que le nombre de configurations produites est une fonction exponentielle du nombre de nœuds, cela limite la taille des réseaux vérifiables.

Pour la solution 2 (Figure 8.3), le nombre de configurations accessibles est 3 : la configuration initiale est *(initial, initial, initial, initial, initial)*, ensuite la source synchronise le groupe $g1$: *(_, initial, received, received, initial)* (où _ est l'état *committed* de la source), puis le groupe $g2$: *(sent, received, received, received, initial)*. Dans ce cas le nombre de configurations augmente linéairement avec le nombre de groupes que la source doit synchroniser.

Avec la solution 3 (Figure 8.4), le nombre de configurations accessibles est également 3 : la configuration initiale est *(initial, initial, initial, initial, initial)*, ensuite la mise à jour des variables globales par la source mène à *(_, initial, initial, initial, initial)*, puis la synchronisation conduit à la configuration *(sent, received, received, received, initial)*. On a donc un nombre de configurations constant et égal à 3.

Nous observons donc que la façon de modéliser les transmissions *broadcast* influence la taille de l'espace des configurations et donc accentue ou atténue le problème d'explosion combinatoire. Les RCsF sont des réseaux de grandes tailles. Pour passer l'échelle lors de la vérification de tels réseaux, il faut utiliser la méthode la plus efficace en terme de nombre de configurations. La comparaison des espaces de configurations des 3 cas nous indique que la solution 1, proposée par [Wibling et al., 2004] et [Tschirner et al., 2008], doit être évitée car elle induit l'espace de configurations le plus important (de plus, son augmentation est exponentielle avec le nombre de nœuds). Les deux autres méthodes induisent un nombre de configurations équivalent, cependant, avec la solution 2 l'augmentation du nombre de configurations est linéaire avec le nombre de groupes auxquels la source appartient, alors qu'avec la solution 3 le nombre de configurations générées est constant. De plus, dans la solution 2, la topologie est directement encodée dans les TA sous forme de transitions et variables de synchronisation, cela ne permet pas de modéliser des changements topologiques (car il faudrait changer dynamiquement la définition des TA, ce qui n'est pas possible). La solution 3 en revanche permet de modéliser les changements topologiques en modifiant la matrice de connectivité.

Nous concluons donc que la solution 3 doit être préférée aux autres pour la modélisation et vérification des protocoles de RCsF. Dans la section suivante, nous modélisons le protocole GRAB avec les 3 solutions de représentation des transmissions *broadcast*. Cela nous permet de conforter nos conclusions dans le cadre d'une étude d'un protocole pour RCsF avec UPPAAL.

8.2.3 Comparaison lors de la vérification d'un protocole pour RCsF

Nous choisissons de modéliser le protocole de routage pour RCsF GRAB [Ye et al., 2005a] présenté dans la section 2.1.2 du Chapitre 2, car son fonctionnement

est simple, cela nous permet de nous concentrer sur les méthodes de modélisation des transmissions *broadcast* plutôt que sur la modélisation du protocole en lui-même. Le but de cette section est de vérifier les résultats obtenus avec les modèles minimalistes lors de la vérification d'un protocole pour RCsF de la littérature.

Nous rappelons d'abord brièvement le fonctionnement de GRAB : lors de l'initialisation du réseau, chaque nœud récupère la distance qui le sépare du puits en nombre de sauts (construction du gradient). Les paquets de données sont routés de la manière suivante : quand un nœud émet un paquet, les nœuds qui ont un gradient inférieur à l'émetteur peuvent relayer le paquet (le choix du relayeur dépend de la technique d'élection utilisée, il peut être aléatoire, basée sur la force du signal reçu, de l'énergie, etc). On note qu'il est possible qu'il y ait plusieurs relayeurs, ce qui augmente la fiabilité mais diminue la durée de vie du réseau. Dans cette section nous modélisons GRAB de façon à ce qu'il n'y ait qu'un seul nœud relayeur du paquet (la première réémission du paquet empêche les autres) pour avoir le modèle le plus simple possible (le but ici est de comparer les méthode de modélisation et non d'évaluer GRAB).

Nous produisons trois modèles de GRAB, un pour chaque façon de modéliser les transmissions *broadcast*. Nous vérifions que le délai de bout en bout est inférieur à une borne pour des topologies allant de 5 à 20 nœuds avec un pas de 1 (pour chaque nombre de nœuds, on génère 10 topologies aléatoires connectées).

Vérification de la connectivité après synchronisation

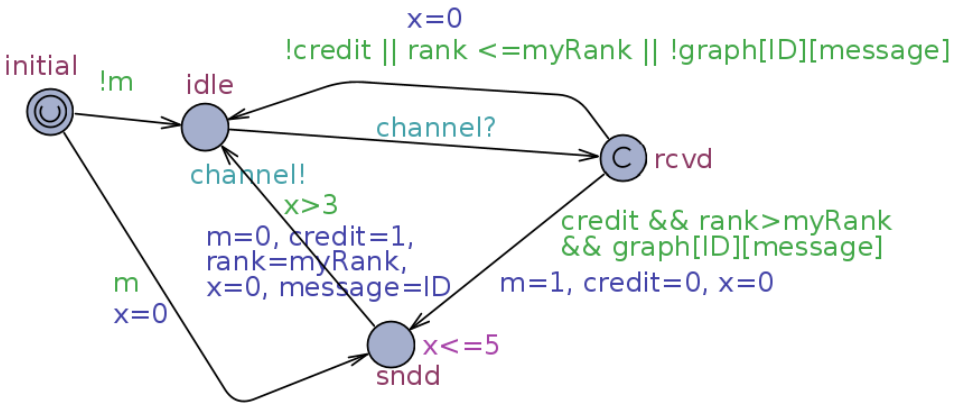


FIGURE 8.5 – TA modélisant GRAB dans le cas où la connectivité est vérifiée après la synchronisation

La Figure 8.5 représente le TA d'un nœud dans le cas où la connectivité est vérifiée après la synchronisation. Au départ, tous les nœuds sont dans l'état `initial`. Si un nœud a un message à transmettre, il passe dans l'état *sndd*, sinon il passe dans l'état `idle` et attend une synchronisation sur la variable canal *broadcast* `channel`. Un nœud dans l'état *sndd* attend entre 3 et 5 unités de temps avant de prendre la transition de synchronisation. Quand il prend cette transition, cela met à jour plusieurs variables : le gradient (`rank`) de l'émetteur, son `ID` et le crédit du message (cette variable définit le nombre de relayeurs possibles pour un paquet, il est ici fixé à 1). L'émetteur et

les récepteurs se synchronisent sur la variable canal `channel`. Les récepteurs passent dans l'état `rcvd`; Si un récepteur est connecté à l'émetteur, qu'il reste du crédit (`credit`> 0) et que sa variable `myRank` (gradient du nœud) est inférieure au `rank` du paquet, le nœud récepteur prend la transition vers l'état `sndd` pour relayer le paquet, sinon la transition vers `idle` est prise (cela décrémente la variable `credit`).

Pour automatiser la production des modèles à vérifier, nous avons développé un outil[†] en Python qui prend en entrée la topologie du réseau et qui produit le modèle UPPAAL à vérifier.

Synchronisation des groupes

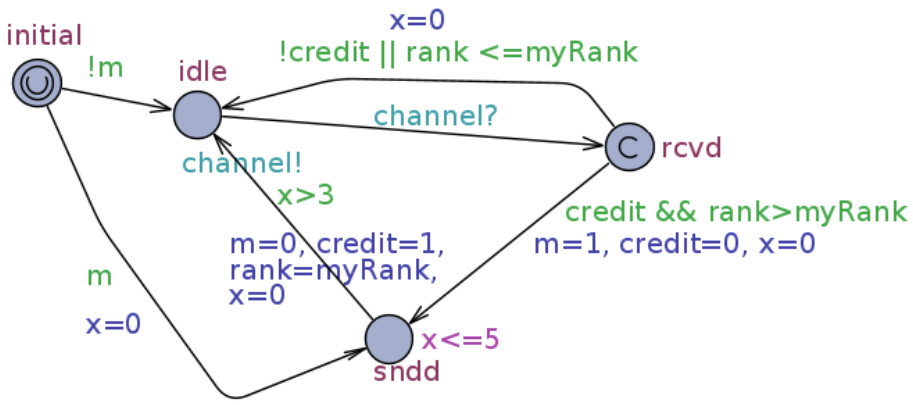


FIGURE 8.6 – TA "de base" modélisant GRAB dans le cas de la synchronisation par groupes

La Figure 8.6 représente le TA "de base" modélisant GRAB dans le cas de la synchronisation par groupes, il devra être adapté à chaque nœud en fonction des groupes auxquels le nœud appartient. Un nœud appartenant à plusieurs groupes possède plusieurs transitions entre `idle` et `rcvd`, une par groupe auquel il appartient, de même il a une suite de transitions et d'états *committed* entre `sndd` et `idle`. La Figure 8.7 est un exemple de TA d'un nœud qui appartient à deux groupes (notés $g1$ et $g3$). On remarque que contrairement au TA de la Figure 8.5, la connectivité n'est pas vérifiée par une garde car un nœud se synchronise avec un émetteur seulement s'ils sont voisins (et donc ont un groupe en commun).

Pour déterminer à quels groupes appartient un nœud et produire son TA, nous avons développé un outil[‡] en Python qui prend en entrée la topologie du réseau et qui donne le modèle UPPAAL du réseau en sortie. Les groupes sont déterminés grâce à l'algorithme de Bron-Kerbosch [Bron and Kerbosch, 1973]. Pour chaque nœud, notre outil crée un TA correspondant au TA "de base" de la Figure 8.6 et y ajoute les

†. Le code de l'outil est visible à cette adresse : `http://perso.citi.insa-lyon.fr/amouradia/code/verification/checkConnectivity.py`

‡. Le code de l'outil est visible à cette adresse : `http://perso.citi.insa-lyon.fr/amouradia/code/verification/groups.py`

synchronisations correspondantes à ses groupes (la Figure 8.7 est un exemple produit par cet outil).

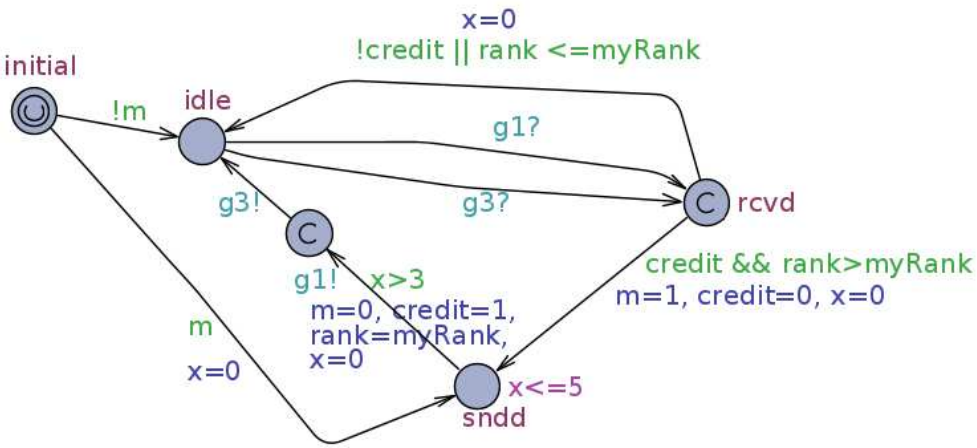


FIGURE 8.7 – Exemple de TA modélisant un nœud GRAB dans le cas de la synchronisation par groupe. Dans cet exemple, le nœud appartient aux groupes $g1$ et $g3$.

Vérification de la connectivité avant synchronisation

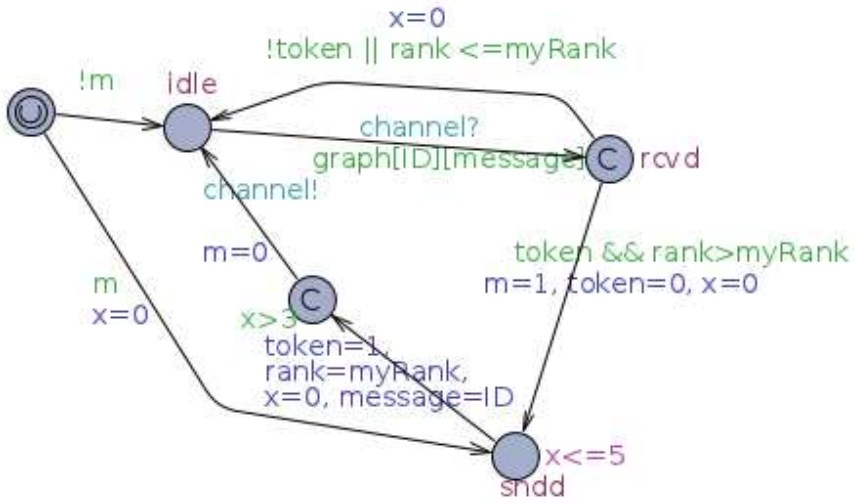


FIGURE 8.8 – TA modélisant GRAB dans le cas où la connectivité est vérifiée avant la synchronisation

La Figure 8.8 représente le TA modélisant GRAB dans le cas où la connectivité est vérifiée avant la synchronisation. Un nœud qui a un paquet à envoyer accède à l'état

`sndd`, il peut y rester entre 3 et 5 unités de temps. Ensuite, il prend la transition vers l'état *committed* (qui n'existe pas avec dans le cas où la connectivité est vérifiée après synchronisation), cette transition met à jour les variables globales. Enfin, il prend la transition de synchronisation vers l'état `idle`. Les nœuds qui sont dans l'état `idle` au moment de la transition ne se synchronisent que s'ils sont voisins de l'émetteur.

Nous utilisons dans ce cas le même outil que pour le cas de la vérification après synchronisation pour générer le modèle UPPAAL (seule la description du TA change).

Performances de la vérification formelle

Pour évaluer les performances de la vérification formelle des 3 modélisations, nous nous intéressons à la taille de l'espace des configurations générées par UPPAAL ainsi qu'à la durée de la vérification.

Nous évaluons ces paramètres sur des topologies de 5 à 20 nœuds (plus un puits par topologie), pour chaque nombre de nœuds 10 topologies sont générées aléatoirement (sélection uniforme des coordonnées (x,y)). Nous vérifions des scénarios pour lesquels un seul paquet est émis. Le nœud émetteur est choisi de manière aléatoire uniforme parmi les nœuds les plus éloignés du puits. La propriété temporelle vérifiée est une propriété de sûreté : "Le paquet a toujours un délai de bout en bout supérieur à $3 \times N$ unités de temps et inférieur ou égal à $5 \times N$ unités de temps", avec N le nombre de sauts que doit effectuer le paquet pour atteindre le puits. Elle est décrite en format UPPAAL dans l'Equation 8.1 :

$$\texttt{A[] n0.m imply (z} > \texttt{N} * 3 \texttt{ and z} \leq (\texttt{N} + 1) * 5) \tag{8.1}$$

`A[]` signifie que la propriété doit être vraie dans toutes les configurations accessibles, `n0.m` signifie que le puits a reçu le paquet, `N` est le nombre de sauts qui sépare l'émetteur du puits et `z` est une horloge qui est mise à zéro lors de l'émission du paquet (le détail du langage d'expression des propriétés de UPPAAL est disponible dans [Gerd Behrmann and Larsen, 2004]). Les valeurs 3 et 5 viennent du modèle : un nœud attend entre 3 et 5 unités de temps avant de relayer un paquet.

On observe sur la Figure 8.9, le nombre de configurations moyen en fonction du nombre de nœuds (la moyenne étant faite sur 10 vérifications avec des topologies différentes) avec plus ou moins deux fois l'écart type. L'échelle de l'axe des ordonnées est logarithmique. Dans la section 8.2.2 nous avons conclu que seule la solution 1 induit une augmentation exponentielle de l'espace de configurations avec l'augmentation du nombre de nœuds, or nous constatons sur la Figure 8.9 que tous les modèles souffrent d'une augmentation exponentielle. Cette augmentation exponentielle n'est pas due (pour les solutions 2 et 3) à la modélisation des transmissions *broadcast*, mais à des entrelacements générés par le passage de l'état `initial` à `idle` ou `sndd`, ou bien encore au choix du relayeur parmi les nœuds voisins de l'émetteur (passage de l'état `rcvd` à `idle` ou `sndd`).

On constate tout de même que la solution qui induit le plus grand nombre de configurations est celle où la connectivité est vérifiée après la synchronisation. Cela est dû à l'entrelacement produit par la vérification de la connectivité comme expliqué dans la

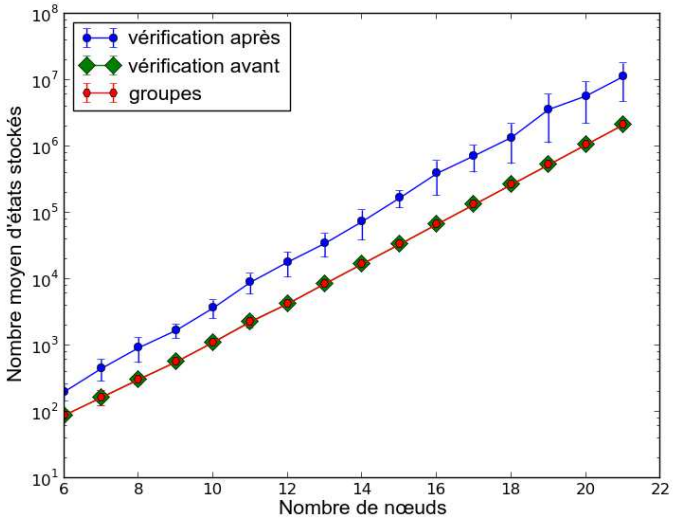


FIGURE 8.9 – Nombre de configurations moyen stocké pendant le *Model Checking* en fonction du nombre de nœuds (les courbes des cas de vérification avant et des groupes sont superposées)

section 8.2.2. les deux autres méthodes ont des performances similaires, leurs courbes sont presque parfaitement superposées. Dans la section 8.2.2 nous avions conclu que le cas de la synchronisation des groupes devrait générer des espaces de configurations plus importants que le cas où la vérification est faite avant la synchronisation, car l'espace de configurations augmente linéairement avec l'augmentation du nombre de groupes de l'émetteur. Cependant, dans les topologies vérifiées, le nombre de groupes auxquels un nœud appartient reste suffisamment faible pour ne pas causer d'augmentation significative de la taille de l'espace des configurations. On note que l'écart type du cas où la connectivité est vérifiée après la synchronisation, bien qu'important, ne recouvre pas ceux des autres cas. On retrouve donc bien les résultats de la section 8.2.2 : la vérification après synchronisation induit un espace de configurations plus important que les deux autres solutions.

La Figure 8.10 représente la durée moyenne du *Model Checking* (la moyenne étant faite sur 10 vérifications avec des topologies différentes en utilisant la commande système `time`) avec plus ou moins deux fois l'écart type, l'échelle de l'axe des ordonnées est logarithmique. On constate que les courbes pour des durées inférieures à une seconde ne sont pas strictement monotones, cela dû à l'impact de l'exécution d'autre processus sur la machine sur laquelle les résultats ont été obtenus. Cet impact diminue pour des durées de *Model Checking* plus importantes. On retrouve avec ces courbes les observations de la Figure 8.9, car la durée d'exploration de l'espace de configurations dépend de sa taille : plus l'espace est grand, plus il faut de temps pour l'explorer (la complexité temporelle du *Model Checking* est linéaire par rapport à la taille de l'espace de configurations [Baier and Katoen, 2008]).

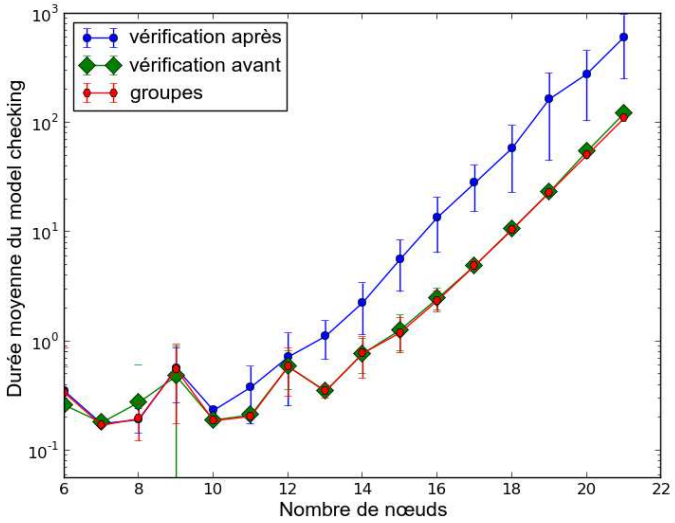


FIGURE 8.10 – Durée moyenne du *Model Checking* en fonction du nombre de nœuds

8.2.4 Conclusions de l'étude

Cette étude est motivée par le fait que plusieurs articles de la littérature utilisent différentes techniques de modélisation des transmissions *broadcast* : cela nous pousse à nous demander si ces techniques sont équivalentes, notamment en termes d'espace de configurations, car l'explosion combinatoire de l'espace des configurations est une des limitations de l'applicabilité du *Model Checking* aux RCsF. Le but de cette étude est donc d'évaluer et de comparer la taille de l'espace des configurations induit par les 3 techniques de modélisation des transmissions *broadcast* recensées dans la littérature. En utilisant des modèles minimalistes qui ne représentent que ce type de transmissions, nous déduisons que celui qui produit le plus petit espace de configurations est le modèle où la connectivité est vérifiée avant la synchronisation, puis vient le modèle de la synchronisation de groupes et le modèle où la connectivité est vérifiée après la synchronisation est celui qui induit le plus grand espace de configurations. Ces observations sont vérifiées par la modélisation du protocole GRAB avec les 3 techniques de transmission et l'étude des performances du *Model Checking* (taille de l'espace des configurations, durée) lors de la vérification avec UPPAAL.

Nous concluons que la technique de modélisation où la connectivité est vérifiée avant doit être préférée pour la vérification des protocoles de RCsF. Non seulement elle induit l'espace des configurations le plus petit parmi les trois étudiés, mais en plus la représentation de la topologie par une matrice d'adjacence permet d'avoir un seul modèle TA pour tous les nœuds du réseau (avec les groupes, les nœuds ont des TA spécifiques) et de modéliser les changements de topologie (modifications de la matrice).

Dans la suite de ce chapitre nous nous intéressons à la modélisation du lien radio non-fiable pour la vérification des RCsF, puis nous proposons une méthode hybride qui

allie *Network Calculus* et *Model Checking* pour pouvoir vérifier des réseaux de grandes tailles tout en vérifiant de manière exhaustive les comportements des protocoles.

8.3 Prise en compte des liens non-fiables lors de la vérification

Grâce à l'étude de la section précédente nous sommes maintenant capables de modéliser efficacement la nature *broadcast* des transmissions sans fil. Cependant, un des aspects les plus importants des transmissions sans fil est le manque de fiabilité de la liaison radio qui rend la réception des paquets aléatoire. Dans la section 7.3 du chapitre 7, nous avons évoqué les limitations des outils qui permettent la vérification de modèles à la fois temporels et probabilistes (les PTA). En effet, PRISM est, à notre connaissance, le seul outil mature pour le *Model Checking* probabiliste. Cependant, il souffre de nombreuses restrictions dues aux techniques de vérification des PTA et à son langage de description de spécifications. UPPAAL, comme on l'a vu dans la section 7.4 a été utilisé avec succès pour modéliser et vérifier des protocoles pour RCsF sans prise en compte du lien non-fiable. Dans cette section, nous proposons une technique qui permet de vérifier formellement les protocoles pour RCsF en prenant en compte l'aspect probabiliste du lien non-fiable et qui est indépendante de l'outil de vérification utilisé. Cette méthode se base sur le constat que la fluctuation des liens (apparition et disparition) implique des changements topologiques qui ont une probabilité quantifiable de se produire : on peut donc, à partir de la position des nœuds estimer quelles topologies logiques sont possibles et avec quelles probabilités associées. Ensuite l'utilisation d'un outil de vérification formelle permet de vérifier si une propriété du protocole étudié est vraie sur chacune des topologies. En sommant les probabilités des topologies où la propriété est vérifiée, on obtient la probabilité que le protocole fonctionne correctement.

La méthode fonctionne de la manière suivante :

- Nous générons toutes les topologies possibles avec leur probabilité associée.
- Pour chaque topologie, le protocole étudié est vérifié.

Dans cette section, nous détaillons ce processus et présentons un exemple d'application de la méthode avec le protocole f-MAC [Roedig et al., 2006], protocole temps-réel dur pour RCsF. En comparant les résultats de validation avec ceux de simulations nous montrons que notre méthode n'est pas trop pessimiste sur l'estimation de la probabilité de vérification de la propriété.

8.3.1 Probabilités d'existence et génération des topologies possibles

Dans cette section, nous décrivons comment à partir de la probabilité de réception d'un paquet, nous déterminons la probabilité d'existence d'une topologie. Une topologie est définie comme un graphe $G = (V, E)$ avec V un ensemble de sommets et $E \subseteq V \times V$ un ensemble d'arêtes. A chaque sommet est associée une position (x, y) sur

Symbole	Description
T	Ensemble de topologies
T'	Ensemble de topologies à vérifier
M	Modèle du protocole
G	Graphe de topologie
E	Ensemble d'arêtes
P_{E_i}	Probabilité associée à l'arête E_i
V	Ensemble de sommets
G_b	Topologie de base
p	Propriété à vérifier
P_{pv}	Probabilité que la propriété soit vraie
$P_{topo}(j)$	Probabilité de la topologie j

TABLE 8.1 – Notations

un plan. Chaque arête peut être marquée comme "active" ou "non-active" avec une probabilité associée. Une arête "active" correspond à des transmissions radio réussies sur le lien radio correspondant, la probabilité de son existence correspond donc à la probabilité qu'un paquet émis sur ce lien soit reçu (cette probabilité est donnée par l'Equation 3.8 : $P_{cr}(d) = 1 - PER(d)$ avec d la distance émetteur-récepteur). On peut noter que la probabilité qu'une arête soit "active" est valable pour un paquet (car la topologie peut changer entre 2 paquets : le premier est reçu et pas le second ou inversement) et dépend du nombre de retransmissions. La probabilité $P(E_i)$ associée à une arête, si elle est "active", est donc $P_{E_i} = P_{cr\_ret}(d_i)$ avec d_i la longueur de l'arête, elle est "non-active" avec la probabilité complémentaire $P_{E_i} = 1 - P_{cr\_ret}(d_i)$. La probabilité de réception d'un paquet en fonction de la distance $P_{cr\_ret}(d_i)$ est issue du modèle de propagation *log-normal shadowing* et exprimée dans la section 3.2.1 du Chapitre 3 : $P_{cr\_ret}(d_i) = 1 - (1 - P_{cr}(d_i))^{K+1}$. Les notations utilisées dans cette section ainsi que la suivante sont définies dans le Tableau 8.3.1.

On note que pour construire notre solution, nous considérons que les liens radios sont symétriques, ce qui n'est pas obligatoirement le cas en réalité dans les RCsF [Heurtefeux et al., 2012]. Cet aspect est discuté dans la section 8.3.3.

La probabilité de la topologie j est donnée par :

$$P_{topo}(j) = \prod_{i=1}^{|E|} P_{E_i} \tag{8.2}$$

Les topologies possibles sont générées à partir d'une topologie de base qui est un graphe avec toutes ses arêtes marquées comme actives. La génération se fait en marquant des sous-ensembles des arêtes comme "non-actives" et d'autres comme actives, tous les cas possibles sont générés : cela produit $2^{|E|}$ topologies (chaque arête peut être active ou non).

L'Algorithme 8.3.1 permet de générer toutes les topologies possibles à partir d'une topologie de base G_b. L'ensemble des topologies est initialisé sur la ligne 1, au début il ne contient que G_b. Les topologies sont générées à l'aide d'une fonction récursive

Algorithm 8.3.1 Algorithme de génération des topologies

Require: topologie de base G_b

```
1: T = {G_b}                          ▷ initialisation de l'ensemble des topologies
2: GÉNÉRER_TOPOLOGIES(G_b)
3: function GÉNÉRER_TOPOLOGIES(G)
4:     for E_i ∈ E marquée comme "active" do
5:         marquer E_i comme "non-active"
6:         if G ∉ T then
7:             add G to T
8:             GÉNÉRER_TOPOLOGIES(G)
9:         end if
10:        marquer E_i comme "active"
11:    end for
12: end function
```

GÉNÉRER_TOPOLOGIES(G) qui prend en paramètre une topologie. Cette fonction boucle sur les arêtes de G, chaque arête "active" est marquée "non-active", la nouvelle topologie est ajoutée à T et la fonction est rappelée sur cette topologie, après le retour de l'appel récursif l'arête courante est de nouveau marquée comme "active".

Si une propriété est vraie sur toutes les topologies, alors la probabilité que cette propriété soit vraie est égale à 1, ceci est garanti par le théorème 8.3.1.

Théorème 8.3.1 $\sum_{j=1}^{2^{|E|}} P_{topo}(j) = 1$

Preuve Considérons une arête E_m, elle est "active" avec la probabilité $P_{cr\_ret}(d_m)$ et "non-active" avec la probabilité $1 - P_{cr\_ret}(d_m)$. Posons A l'ensemble des topologies sans E_m et S la somme des probabilités des topologies de A. Maintenant, ajoutons E_m à toutes les topologies de A, pour être exhaustif il faut ajouter la version "active" et la version "non-active". Cela double le nombre de topologies de A, on peut alors écrire la nouvelle somme S' des probabilités des topologies de A comme suit :

$$\begin{aligned} S' &= \sum_{j \in A}^{2^{|E|}} P_{topo}(j) \\ &= P_{cr\_ret}(d_m) \cdot S + (1 - P_{cr\_ret}(d_m)) \cdot S \\ &= S \end{aligned} \tag{8.3}$$

on a donc $S = S'$ c'est-à-dire que l'addition d'une arête ne change pas la somme, si on applique ce raisonnement à toutes les arêtes on conclut que $S = S_1$ avec S_1 la somme des probabilités des topologies pour un graphe d'une arête. Or $S_1 = P_{cr\_ret}(d_1) + (1 - P_{cr\_ret}(d_1)) = 1$ donc $S = 1$. ∎

La Figure 8.11 illustre une topologie de base simple (3 nœuds et 2 liens) et ses topologies dérivées. La topologie de base est la numéro 1. On peut, par exemple calculer la probabilité de la topologie 3 : on a E_1 "active" donc $P_{E_1} = P_{cr\_ret}(d_1)$ et E_2 "non-active" donc $P_{E_2} = 1 - P_{cr\_ret}(d_2)$ et d'après l'Equation 8.2 $P_{topo}(3) = P_{E_1} \times P_{E_2}$.

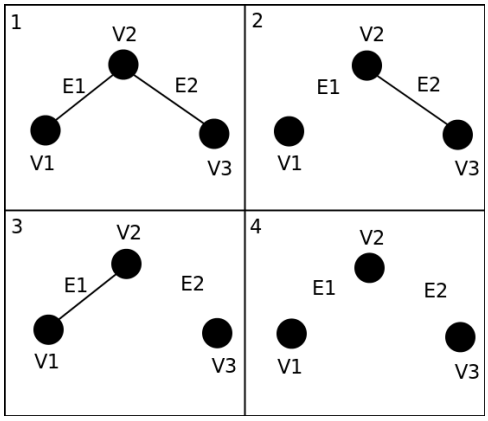


FIGURE 8.11 – Exemple avec 3 nœuds et 2 liens

On note que pour être vraiment exhaustif dans la vérification des topologies possibles, il faut que la topologie de base soit une clique : il existe toujours une probabilité non-nulle que deux nœuds puissent communiquer entre eux. Cependant, cela oblige à vérifier des topologies qui ont une très faible probabilité d'exister (quand les liens sont très longs). Il est plus intéressant de se concentrer sur les topologies qui ont une forte probabilité d'exister, car si un problème de comportement est détecté sur ces topologies, il a de fortes chances d'apparaître souvent dans la réalité. Pour écarter les topologies qui ont des probabilités faibles, on génère la topologie de base en utilisant un seuil de probabilité pour les liens (comme dans le Chapitre 3 avec l'Equation 3.10 : $P_{cr}(R) = P_{th}$, R étant donc la longueur maximale d'un lien).

Dans la section suivante, nous détaillons le fonctionnement de la méthode de vérification.

8.3.2 Méthode de vérification

Pour obtenir la probabilité qu'une propriété est vraie, nous proposons de vérifier le modèle du protocole étudié sur chaque topologie et de sommer les probabilités des topologies pour lesquelles la propriété est vraie. On rappelle que l'existence d'une topologie et sa probabilité d'existence n'ont de sens que pour une durée limitée (cf. section 8.3.1), il faut donc limiter la durée de la vérification (vérification des comportements sur une durée prédéterminée). Comme nous l'avons observé dans le Chapitre 2, les protocoles pour RCsF ont des comportements réguliers : sous un régime de trafic donné, ils répètent les mêmes actions tous les cycles d'endormissement (émettre un paquet, en relayer un, etc). Nous proposons donc de contraindre la vérification à la durée d'un cycle d'endormissement, on peut alors vérifier des propriétés du type : "Tous les paquets avancent d'un saut vers le puits durant un cycle d'endormissement". Si la propriété est vrai pour un cycle d'endormissement et que le protocole a le même comportement pour tous les cycles, alors la propriété est vérifiée.

L'Algorithme 8.3.2 résume le processus de vérification. Il prend en entrée l'ensemble des topologies générées par l'Algorithme 8.3.1, le modèle du protocole à vérifier (qui dépend de l'outil de vérification utilisé) et la propriété à vérifier. D'abord, la

Algorithm 8.3.2 Processus de vérification

Require: ensemble des topologies T, modèle du protocole M, propriété à vérifier p
  Calculer les probabilités des topologies
  Selectionner les topologies à vérifier T'
  for $T_j \in T'$ **do**
    Vérifier p sur M avec T_j
    if p est vraie **then**
      $P_{pv} \leftarrow P_{pv} + P_{topo}(j)$
    end if
  end for

probabilité des topologies est calculée grâce à l'Equation 8.2. Ensuite, certaines topologies sont exclues du processus de vérification : comme mentionné dans la section précédente, l'ensemble T contient $2^{|E|}$ topologies, or il est possible de savoir que la propriété n'est pas vérifiée sur certaines topologies sans en effectuer la vérification formelle. Par exemple, si la propriété est : "tous les paquets atteignent le puits en moins de 5s", on peut alors éliminer toutes les topologies non connectées (dans l'exemple de la Figure 8.11, seule la topologie 1 serait vérifiée). Les topologies sélectionnées pour la vérification sont stockées dans T'. Pour chaque topologie sélectionnée, une vérification est exécutée, si la propriété est vraie, sa probabilité associée est ajoutée à P_{pv}, probabilité que la propriété soit vraie.

Le choix de l'outil de vérification est laissé à l'utilisateur. L'outil doit être capable de fournir une réponse positive ou négative à la question : "le modèle (incluant la topologie) respecte-t-il la propriété ?". On note que si la méthode de vérification est automatique (*Model Checking*), tout le processus de vérification peut être automatisé.

La méthode proposée permet de détecter les topologies qui ne sont pas gérées par le protocole vérifié, en ce sens elle s'inspire des travaux de [Fehnker et al., 2007] (qui vérifie LMAC sur toute les topologies connectés de 4 et 5 nœuds), mais en plus elle donne la probabilité d'apparition des "mauvais" cas. Avec cette information, lors de la conception d'un protocole, on peut choisir d'adapter le protocole ou non à ces cas en fonction de la fiabilité visée, sachant que prendre en compte de nouveaux cas peut avoir un coût en termes d'énergie ou bien de délai de bout en bout.

Applicabilité de la méthode

Comme mentionné dans la section précédente, la probabilité d'existence des topologies est valide pour une durée limitée dans le temps (nous proposons de la limiter à un cycle d'endormissement). On peut donc se demander si cette méthode est capable de capturer l'aspect séquentiel des communications de bout en bout dans les RCsF et donc si cette méthode peut être appliquée aux protocoles de routage. Pour contourner cette limitation apparente, nous proposons de vérifier les protocoles de routages en deux étapes : d'abord nous calculons, grâce au modèle de trafic, le pire cas de congestion au niveau de chaque nœud (le nombre maximal de paquet qu'un nœud a dans sa file d'attente) ; ensuite, la seconde étape consiste à vérifier la propriété suivante :

"Dans le pire cas de congestion, tous les paquets font au moins un saut en direction du puits durant un cycle d'endormissement". Le fait que tous les paquets se rapprochent du puits dans un temps borné doit être assuré par le protocole de routage (si le temps d' accès au médium et le nombre de sauts sont bornés, le délai de bout en bout l'est aussi). La méthode que nous proposons est donc applicable aux protocoles MAC et routage, mais la propriété doit être vérifiée sur une plage de temps qui correspond à la durée de validité de la topologie (déterminée par le nombre de retransmissions d'un paquet).

8.3.3 Cas d'étude : vérification temporelle de f-MAC

Dans cette section nous appliquons la méthode proposée pour vérifier le protocole f-MAC décrit dans la section 2.2.1 du Chapitre 2. Nous détaillons plus précisément le fonctionnement du protocole, nous appliquons la méthode de vérification et nous comparons les résultats avec des simulations pour estimer le pessimisme de notre méthode (sous-estimation de la probabilité que la propriété soit vraie).

Détails de f-MAC

Nous choisissons de vérifier f-MAC car il fournit un accès en temps borné au médium dans le cas d'un lien radio où les pertes de paquets sont seulement dues aux collisions [Roedig et al., 2006]. Il est donc intéressant d'étudier f-MAC avec des liens non-fiables pour déterminer si cette propriété est toujours respectée.

f-MAC est un protocole MAC asynchrone sans cycle d'endormissement : les nœuds ne maintiennent pas de synchronisation, mais écoutent en permanence le canal lorsqu'ils n'émettent pas. L'approche utilisée pour les transmissions est appelée transmission par *framelet* : le paquet de donnée est envoyé plusieurs fois de manière périodique, chaque occurrence du paquet est appelée *framelet*. Dans le cas de f-MAC, les retransmissions ne servent pas la fiabilité, mais permettent de résoudre les collisions, pour cela il faut paramétrer les *framelets* en suivant les règles suivantes :

1. La taille d'une *framelet* est : $d = \delta/2$ où δ est l'unité de base de f-MAC.
2. Le nombre de *framelets* est : $r = N_n$ avec N_n le nombre de nœuds.
3. La période d'émission des *framelets* du nœud i, $t_i = k_i \times \delta$ doit satisfaire :

$$k_i \cdot (r-1) < LCM(k_i, k_j) \quad \forall k_i < k_j \quad 1 \leq i, j \leq N_n$$

   avec $LCM(k_i, k_j)$ le plus petit commun multiple de k_i et k_j
4. Les nœuds doivent attendre $t' = (k_{max} \cdot (r-1) + 1).\delta$ après la dernière *framelet* avant d'envoyer un autre paquet (avec $k_{max} = \max_{1 \leq i \leq N_n}\{k_i\}$).

A partir de ces règles, on peut déduire que la pire durée de transmission pour le nœuds i, notée T_i, est :

$$T_i = (r-1) \cdot t_i + t' \tag{8.4}$$

Les auteurs de [Roedig et al., 2006] prouvent, qu'en suivant ces règles de choix des périodes de *framelets* et sous hypothèse de lien fiable, la réception d'au moins une

framelet est garantie (une *framelet* ne subira pas de collision). La pire durée de transmission pour tous les nœuds est déduite de l'Equation 8.4 : $T_{max} = \max_{1 \leq i \leq N_n}\{T_i\}$.

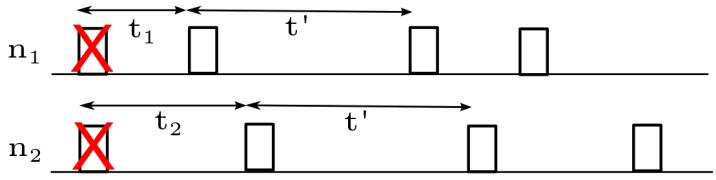
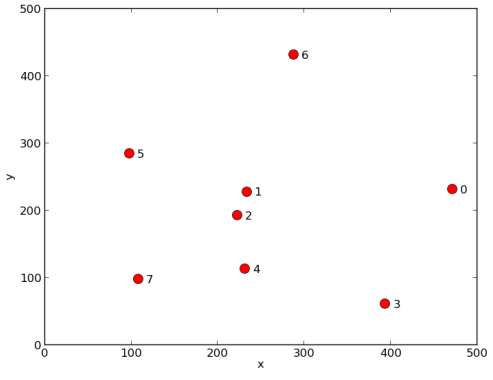


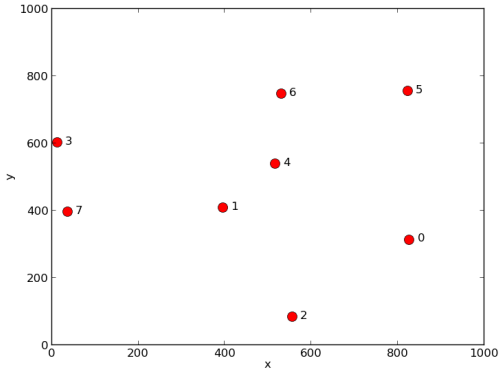
FIGURE 8.12 – Exemple de f-MAC avec 2 nœuds

La Figure 8.12 représente un exemple avec deux nœuds. Dans ce cas, la règle numéro 2 impose l'émission de deux *framelets* par paquet, dans cet exemple les deux nœuds ont un paquet à envoyer. La première *framelet* du premier paquet est perdue à cause d'une collision, mais la seconde *framelet* est reçue car les nœuds n_1 et n_2 ont des périodes différentes. On constate bien qu'au moins une *framelet* par paquet est reçue.

Pour la vérification et la simulation, nous produisons deux topologies de base de 8 nœuds qui sont tous connectés directement (ce sont des cliques). La topologie de base A est représentée sur la figure 8.13(a) (les liens ne sont pas représentés pour que la figure reste lisible), les nœuds sont placés de manière aléatoire uniforme et de telle façon que la distance maximale entre deux nœuds soit de 500 unités de longueur. Pour la topologie B, illustrée par la figure 8.13(b), la distance maximale est de 1000 unités de longueur, dans ce cas, les liens sont plus longs et donc les probabilités de réception plus faibles.



(a) Topologie de base A, 500m maximum



(b) Topologie de base B, 1000m maximum

FIGURE 8.13 – Représentations des topologies de base utilisées lors des vérifications et simulations de f-MAC.

Le nœud 0 est le puits, pour les vérifications et les simulations, nous utilisons ces topologies mais en faisant varier le nombre de nœuds entre 3 et 8 (les nœuds non utilisés sont considérés comme inactifs). Par exemple, la topologie A-3 est la topologie A avec seulement les nœuds 0, 1 et 2 actifs.

Vérification

Nous appliquons maintenant le processus de vérification décrit dans la section 8.3.2 au protocole f-MAC avec les topologies de base A et B. Pour les raisons évoquées dans la section 7.3 du Chapitre 7, nous modélisons et vérifions f-MAC à l'aide de l'outil UPPAAL.

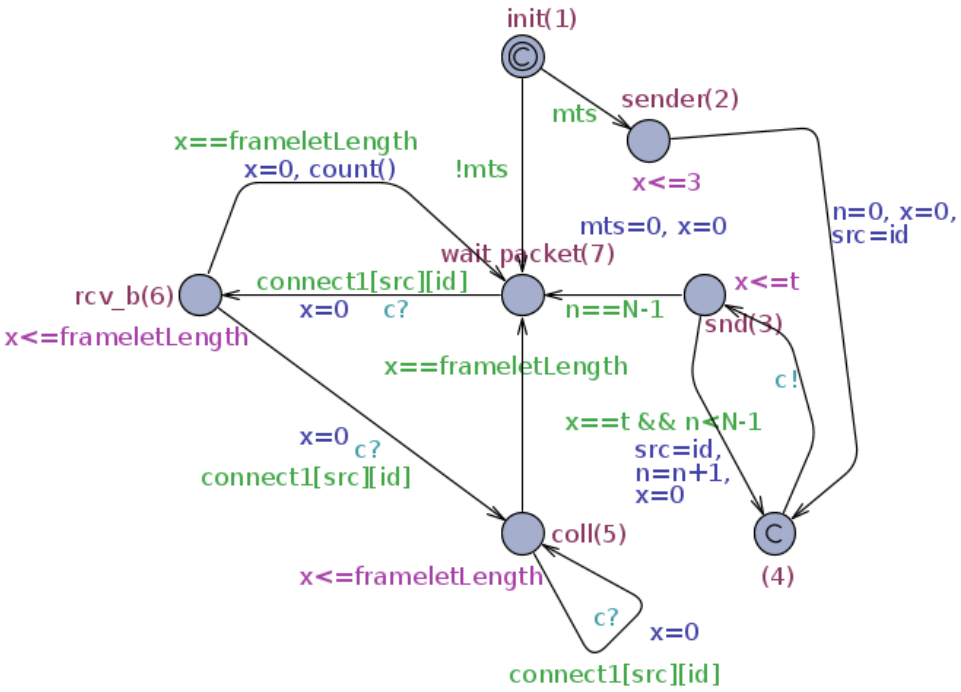


FIGURE 8.14 – Modélisation UPPAAL de f-MAC

La Figure 8.14 représente le modèle UPPAAL d'un nœud f-MAC. Depuis l'état initial `init(1)`, les nœuds qui ont un message à envoyer (variable $mts \geq 1$) passent à l'état `sender(2)`, les autres à l'état `wait packet(7)`. Depuis `sender(2)` un émetteur peut prendre la transition entre 0 et 3 unités de temps, cela permet de désynchroniser les nœuds pour vérifier des cas où les nœuds n'émettent pas leur première *framelet* au même instant : cette désynchronisation induit un nombre de configurations important (on prend la valeur 3 car des valeurs plus importantes engendrent une explosion combinatoire du nombre de comportements). Ensuite, les émetteurs bouclent entre les états `snd(3)` et `(4)` jusqu'à que N_n *framelets* soient envoyées. La période des *framelets* est contrôlée par l'invariant $x <= t$ et la garde $x == t$. L'émission d'une *framelet* est modélisée par l'envoi d'un signal de synchronisation sur la variable canal *broadcast* c : $c!$. Dand nos scénarios les nœuds ont seulement un paquet à envoyer, après N_n

Nombre de nœuds	Nombre de topologies	Nombre de topologies vérifiées	Nombre moyen d'états	Durée (s)		P_{pv}	
				A	B	A	B
3	8	2	74.5	0.30	0.21	0.931	0.746
4	64	8	934	0.32	0.33	0.918	0.368
5	1024	64	10753	15.11	15.22	0.875	0.323
6	1073741824	1024	149546.6	69066.86	69173.25	0.771	0.268

TABLE 8.2 – Résultats de vérification de f-MAC

framelets les nœuds passent dans l'état `wait packet(7)`. Le puits est le nœud 0, il n'a pas de paquet à transmettre, il passe donc directement à l'état `wait packet(7)`. Les nœuds dans cet état ne se synchronisent avec l'émetteur que s'ils sont connectés : on utilise la méthode de modélisation des transmissions où la connectivité est vérifiée avant synchronisation (cf. section 8.2). Si le nœud est connecté à l'émetteur, il passe dans l'état `rcv_b(6)`. Il reste dans cet état pour une durée qui correspond à la durée d'une *framelet*, si pendant cette durée, le nœud reçoit une deuxième framelet (deuxième synchronisation), cela signifie qu'il y a collision entre les *framelets*. Dans ce cas, la *framelet* n'est pas reçue et le nœud passe à l'état `coll(5)`, il reste dans cet état jusqu'à la fin de la deuxième *framelet*, si une autre arrive entre temps, le nœud boucle dans cet état sinon il retourne dans l'état `wait packet(7)`. Si il n'y a pas de collision, la fonction `count()` compte le nombre de paquets reçus par le nœud.

Dans le modèle, la topologie est représentée par la matrice d'adjacence `connect1`. Pour effectuer la vérification, nous avons produit un outil[§] en Python et Shell qui implémente les Algorithmes 8.3.1 et 8.3.2. L'outil intègre aussi les topologies dans les modèles UPPAAL.

Pour le calcul des probabilités des topologies, on remarque que dans le cas de f-MAC, les retransmissions ne sont pas utilisées pour augmenter la fiabilité des transmissions, mais pour accéder au médium en temps borné. Elles ne doivent donc pas être prises en compte dans le calcul des probabilités des topologies, car dans le pire cas, toutes les *framelets* de chaque nœud sauf une, entrent en collision. En prenant en compte le pire cas, on s'assure que les probabilités obtenues par vérification sont des probabilités minimums que la propriété soit vraie (comme on le vérifiera grâce aux résultats de simulations). Les probabilités sont calculées grâces à l'Equation 3.8 (P_{cr_ret}) avec les paramètres du Tableau 3.1 (p.44 section 3.2.1, Chapitre 3) et l'Equation 8.2. f-MAC n'implémente pas de cycle d'endormissement, nous proposons donc de limiter la durée de vérification à la durée maximale de transmission d'un paquet exprimée par l'Equation 8.4.

Le Tableau 8.2 présente les résultats du processus de vérification pour les topologies de base A et B. Le nombre de topologies générées, le nombre de topologies vérifiées, le nombre moyen d'états stockés durant le *Model Checking* et la durée de vérification augmentent de manière exponentielle avec le nombre de nœuds. Pour les

§. Le code de l'outil est visible aux adresses suivantes : `http://perso.citi.insa-lyon.fr/amouradia/code/verification/generateCases.py` et `http://perso.citi.insa-lyon.fr/amouradia/code/verification/checkCases.sh`

deux topologies de base, on observe que la probabilité que la propriété soit vérifiée (c'est-à-dire que tous les paquets soient reçus par le puits avant leur échéance) diminue quand le nombre de nœuds augmente. C'est dû au fait que, lorsqu'il y a plus de nœuds, plus de paquets doivent atteindre le puits et la probabilités qu'ils soient tous reçus est plus faible. La vérification est possible pour des réseaux de 6 nœuds au plus, cette limitation est discutée plus loin dans cette section. Pour la topologie B, la probabilité P_{pv} décroît rapidement entre 3 et 4 nœuds (topologies B-3 et B-4), c'est dû au fait que le nœud 3 est beaucoup plus loin du puits que les nœuds 1 et 2 (la topologie B-4 contient le nœud 3, ce qui n'est pas le cas de B-3), or la probabilité de recevoir un paquet dépend de la distance émetteur-récepteur. Pour la topologie A, on observe aussi ce phénomène mais entre les topologies de 5 et 6 nœuds.

Le protocole f-MAC respecte la propriété pour toutes les topologies vérifiées (i.e. les topologies pour lesquelles tous les nœuds sont connectés au puits). Cela signifie que le protocole f-MAC a un comportement correct, et donc que si les liens étaient fiables la propriété serait respectée avec une probabilité égale à 1.

Ces résultats nous permettent de conclure que le protocole à un comportement sûr avec une probabilité de P_{pv}. C'est une information utile lors de la conception d'un protocole. En effet, si la probabilité est suffisante pour l'application, le système peut être implémenté, sinon la conception doit être modifiée (ajout de retransmissions, modification de la position des nœuds, ajout de nœuds, etc).

Discussion sur la méthode de vérification

Dans cette section, nous discutons des avantages et inconvénients de la méthode de vérification proposée. La limitation principale de notre solution est sa capacité à passer l'échelle. En effet, lors de la vérification de f-MAC, le nombre de nœuds varie seulement de 3 à 6 (et non 8) car la durée de vérification explose pour des réseaux de plus de 6 nœuds. Cela ne vient pas seulement de l'explosion combinatoire de l'espace des configurations, mais aussi de celle du nombre de topologies à vérifier, en effet pour 7 nœuds, 1073741824 cas doivent être vérifiés. On peut noter que le modèle du protocole a une influence sur le passage à l'échelle de la solution. On verra dans le Chapitre 9, qu'en effet des réseaux un peu plus importants peuvent être vérifiés lorsque le protocole est synchrone (on évite alors les configurations issues de la désynchronisation évoquées dans le cas de f-MAC). Une solution pour passer l'échelle est de vérifier d'abord les topologies les plus probables, lorsque la fiabilité requise par l'application est atteinte ou si on constate qu'elle ne peut plus être atteinte, on stoppe la vérification. Cette solution est efficace si la distribution des probabilités est très déséquilibrée : peu de topologies concentrent des probabilités élevées. On note tout de même que le passage à l'échelle est un problème pour toutes les solutions de la littérature, comme mentionné dans la section 7.4 du Chapitre 7, notamment les modèles vérifiés avec PRISM qui ne comportent au maximum que 4 nœuds [Fruth, 2006]. Malgré cette limitation, la technique proposée peut être utilisée dans le cadre de la vérification formelle des BAN (*Body Area Networks*) car ils sont habituellement composés de seulement quelques nœuds (moins de 10, le plus souvent

5 ou 6 [IEEE, 2012]) et les applications médicales requièrent le respect de contraintes temporelles et de fiabilité.

La non prise en compte des liens radio asymétriques est une autre limitation de notre solution. L'ajout de la prise en compte de ce type de liens ne pose, en théorie, pas de problème : il faut considérer des topologies sous forme de graphes orientés. En pratique, cela double le nombre d'arêtes et donc augmente d'une puissance de 2 le nombre de topologies à vérifier, cela limite encore le passage à l'échelle.

Le principal avantage de la solution proposée est que l'évaluation de la probabilité que la propriété soit vraie est indépendante de l'outil de vérification utilisé. Nous utilisons le *model checker* UPPAAL dans notre exemple, mais on peut très bien imaginer d'autres méthodes : par exemple, si on pose le problème de l'accès au canal comme un problème d'ordonnancement (ordonnancement des paquets sur le médium), on peut imaginer vérifier l'ordonnançabilité de chaque topologie (et en déduire la probabilité d'ordonnançabilité).

La solution proposée permet, non seulement de trouver des erreurs de conception de protocoles mais aussi leurs probabilités d'occurrence. Cela permet au concepteur de choisir quelles erreurs corriger en priorité en fonction de leurs probabilités. Si aucune erreur n'est détectée (comme dans le cas de f-MAC), le résultat correspond à la fiabilité du déploiement considéré. On peut alors comparer la fiabilité de plusieurs déploiements en produisant plusieurs topologies de base et en comparant la probabilité que la propriété soit vérifiée dans chaque cas.

Dans la section suivante, nous simulons le protocole f-MAC en utilisant les topologies de base A et B. La probabilité que, lors d'une simulation, tous les paquets arrivent au puits avant l'échéance est évaluée, cela permet d'évaluer le pessimisme de la technique de vérification proposée par rapport aux résultats de simulations.

Simulations de f-MAC

Les simulations sont réalisées à l'aide du simulateur à événements discret WSNet [WSNet, 2009] déjà présenté dans les chapitres de la première partie de ce document (notamment dans la section 4.2.2 du Chapitre 4). Les paramètres de simulation sont les mêmes que pour les vérifications : le modèle de propagation est le modèle *log-normal shadowing*. En revanche, contrairement au cas théorique, la propagation de chaque paquet est simulée. Dans WSNet, la probabilité de recevoir un paquet dépend aussi des interférences avec les autres nœuds : il est donc possible de décoder un paquet même si un nœud interférant est en train de communiquer. Ce n'est pas le cas pour la vérification où des communications simultanées provoquent obligatoirement une collision.

Deux jeux de simulations sont effectués : simulations du cas moyen et simulations du pire cas de collisions de *framelets*. Dans les deux cas, on reprend les mêmes topologies que pour la vérification. Pour chaque nombre de nœuds (de 3 à 8), 10000 simulations sont effectuées. La durée d'une simulation est de $2 \times T_{max}$ (déduit de l'Equation 8.4) et chaque nœud a un paquet à transmettre au puits. Pour les simulations du cas moyen, les nœuds choisissent aléatoirement de façon uniforme une date de début de transmission entre 0 et T_{max} (puis ils émettent périodiquement des

framelets). Cela permet de désynchroniser les nœuds comme pour le modèle de vérification. Pour les simulations du cas pire, une seule *framelet* est émise par les nœuds à des instants différents, cela correspond au cas où une seule *framelet* ne subit pas de collision.

Dans les deux cas, nous calculons le rapport entre le nombre de simulations où tous les paquets respectent leur échéance T_{max} et le nombre total de simulations. Ce rapport est comparé à la probabilité que la propriété soit vraie.

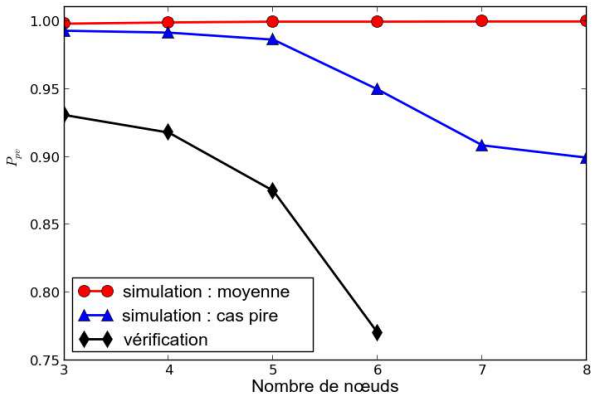


FIGURE 8.15 – Résultats pour la topologie de base A

La Figure 8.15 représente le rapport de simulations où la propriété est vérifiée en fonction du nombre de nœuds pour la topologie de base A. Les résultats des deux cas de simulations et de la vérification sont tracés. On constate d'abord que les résultats de vérification formelle sont inférieurs aux résultats de simulations. Le cas moyen (toujours proche de 1) est très supérieur aux résultats de vérification, c'est dû au fait que nous prenons le pire cas de collisions de *framelets* pour calculer les probabilités de topologies dans le cas de la vérification. Le pire cas de collision est rare dans les simulations. Dans la plupart des cas, plusieurs *framelets* n'entrent pas en collision, cela augmente la probabilité de réception du paquet. De plus, WSNet modélise l'effet de capture [Son et al., 2006] : si une collision survient le nœud se synchronise et décode le paquet dont le signal est le plus puissant, si la puissance est équivalente pour les deux paquets aucun n'est décodé. Dans le modèle de vérification une collision implique forcément la perte des paquets. Cet effet augmente d'autant plus les résultats des simulations du cas moyen.

On observe que les résultats de simulations pour le cas pire de collisions sont au-dessus de ceux des vérifications. Cependant, les courbes ont la même forme et la vérification est conservative mais pas exagérément pessimiste.

La Figure 8.16 représente le rapport de simulations où la propriété est vérifiée en fonction du nombre de nœuds pour la topologie de base B. Dans ce cas encore, les résultats de simulations sont supérieurs à ceux des vérifications. Comme observé dans le cas des vérifications, la probabilité de respect de la propriété décroît fortement entre les topologies de 3 et 4 nœuds car le nœud 3 est très éloigné du puits. En revanche,

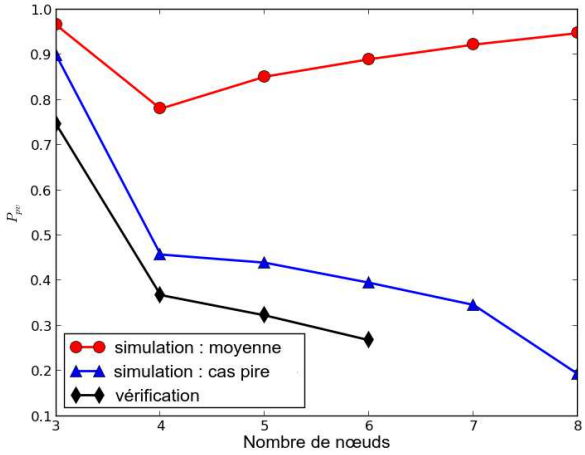


FIGURE 8.16 – Résultats de simulations pour la topologie de base B

pour les simulations du cas moyen de collisions, on constate une augmentation entre les topologies de 4 et 8 nœuds. C'est dû à la règle 2 de f-MAC qui implique d'envoyer plus de *framelets* lorsqu'il y a plus de nœuds, grâce à l'effet capture, cela augmente la probabilité de recevoir un paquet. Avec la topologie de base B également, on constate que les courbes des résultats de simulations du pire cas et ceux de vérification ont la même forme et que la vérification n'est pas exagérément pessimiste.

On constate donc que les résultats de validation sont inférieurs aux résultats de simulations pour les topologies étudiées. On observe que la méthode de vérification proposée permet bien de capturer le pire cas, car les résultats de simulations du pire cas sont très similaires à ceux de vérification. De plus, les résultats de vérification ne sont pas exagérément pessimistes par rapport aux simulations.

8.3.4 Conclusion sur la méthode de vérification avec prise en compte des liens non-fiables

Devant le manque d'outils de vérification qui permettent de représenter et de vérifier les modèles à la fois temporels et probabilistes, nous proposons une méthode qui permet de prendre en compte les liens radios non-fiables pour la vérification des protocoles temps-réel pour RCsF. Pour cela nous générons les topologies possibles du fait de la fluctuation des liens radios et nous leurs assignons des probabilités qui dépendent de la probabilité de succès de transmission sur les liens radios (calculées à partir du modèle de propagation radio, dans notre cas *log-normal shadowing*). La vérification formelle est effectuée sur chaque topologie, la méthode permet donc de détecter les cas où la propriété n'est pas vérifiée ainsi que leurs probabilités d'apparition. La méthode ne repose pas sur un outil de vérification en particulier, l'outil doit seulement répondre positivement ou négativement à la question : "la propriété est-elle vraie sur le modèle du protocole avec une topologie donnée ?". Nous appliquons la méthode proposée au protocole f- MAC, nous utilisons l'outil UPPAAL pour pouvoir

vérifier le comportement temporel du protocole. Les résultats de vérifications capturent bien le pire cas et sont donc plus pessimistes que les résultats de simulations du cas moyen. La simulation du pire cas montre, cependant, que la méthode n'est pas exagérément pessimiste. Nous montrons, par ailleurs, que f-MAC est un protocole temps-réel correct, car nous n'avons pas détecté de fautes provenant du protocole.

La principale limitation de cette méthode est la difficulté du passage à l'échelle. Dans la section suivante, nous proposons une méthode qui allie le *Network Calculus* et le *Model Checking* pour pouvoir passer l'échelle tout en explorant de manière exhaustive les comportements du protocole vérifié.

8.4 Passage à l'échelle : solution hybride *Network Calculus* et *Model Checking*

Comme nous l'avons constaté dans les sections précédentes, le principal inconvénient du *Model Checking* est l'explosion combinatoire de l'espace de configurations et donc l'impossibilité de vérifier de grands systèmes. Nous avons mentionné dans le chapitre 7 que le *Network Calculus* ne souffre pas de ce problème, car les opérateurs mathématiques qu'il utilise pour la composition des nœuds et l'obtention de bornes temporelles ont une complexité calculatoire réduite [Le Boudec and Thiran, 2001]. Cela est possible car le comportement complexe des nœuds est abstrait par des courbes qui représentent les pires cas de service et de trafic du nœud. Dans la littérature [Le Boudec and Thiran, 2001] [Wandeler, 2006] [Schmitt and Roedig, 2005] [Schmitt et al., 2007], ces abstractions sont admises sans preuve formelle de leurs exactitudes. Pour des protocoles au comportement facilement prédictible tel que le TDMA, on peut accepter cette absence de preuve, en revanche pour des protocoles plus complexes tel que RTXP, il est plus difficile d'être absolument sûr du service minimal fourni.

Dans cette section nous proposons une méthode hybride, qui allie *Network Calculus* et *Model Checking* pour être capable, à la fois de passer l'échelle, et de fournir un niveau de confiance plus élevé que *Network Calculus* seul. Pour cela, nous proposons de vérifier que la courbe de service proposée pour le protocole étudié est conforme au modèle du protocole. Nous nous appuyons sur des travaux d'adaptation du *Network Calculus* aux RCsF [Schmitt and Roedig, 2005] [Schmitt et al., 2007] et de correspondance entre les courbes de *Network Calculus* et TA [Lampka et al., 2009].

8.4.1 Description générale de la solution proposée

Le problème d'explosion combinatoire du *Model Checking* apparaît lors de l'augmentation de la taille des réseaux à vérifier : les comportements concurrents des nœuds induisent un espace de configurations trop important pour être contenus dans la mémoire d'une machine et exploré. Le *Network Calculus* n'est pas soumis à ce problème car les comportements concurrents sont abstraits, il sont bornés par des fonctions mathématiques (nommées courbes) dont la définition formelle sera donnée dans la

section 8.4.2. Notre méthode tire parti de cet aspect de *Network Calculus* tout en vérifiant formellement les bornes proposées. Elle consiste, pour chaque nœud, en deux points :

1. Abstraire les interactions du nœud avec le réseau sous forme de courbes de *Network Calculus*.
2. Vérifier par *Model Checking* que le nœud (modèle du protocole) est capable de gérer ces interactions (servir le trafic) en temps borné.

Avec le *Network Calculus*, on peut résumer toutes les interactions d'un nœud avec le reste du réseau par une courbe qui représente les paquets que le nœud reçoit du réseau (ou du fait de détections de phénomènes physiques) et une courbe qui représente les paquets que le nœud émet vers le réseau. Les paquets émis dépendent des paquets reçus et du service que peut fournir le nœud. Notre méthode de vérification hybride consiste à faire l'hypothèse que chaque nœud du réseau fournit un service minimum qui garantit que les événements sont traités dans un temps borné et à vérifier cette hypothèse par *Model Checking* pour chaque nœud. Pour cela nous définissons pour chaque nœud :

- une courbe qui correspond à une borne supérieure sur le taux d'alarmes (détection d'événements) produites par ce nœud ;
- une courbe qui correspond à une borne supérieure sur les alarmes que ce nœud doit relayer (qui viennent des nœuds plus éloignés du puits) ;
- une courbe qui correspond à une borne supérieure sur les délais dus aux accès concurrents au médium. En effet, dans les réseaux sans fil les accès concurrents limitent le service qu'un nœud peut fournir ;
- une courbe qui correspond à l'hypothèse sur le service fourni par le nœud (nombre de paquets qu'il peut émettre dans un temps borné, ce paramètre dépend du protocole étudié) ;
- une courbe qui correspond à une borne supérieure sur les alarmes émises par ce nœud (calculée en fonction des entrées)

Pour le calcul de ces courbes, nous utilisons les travaux [Schmitt and Roedig, 2005] et [Schmitt et al., 2007] qui consistent en une application du *Network Calculus* aux RCsF. Pour calculer les courbes des nœuds, la topologie du réseau est fournie sous forme d'arbre enraciné sur le puits (l'arbre dépend du protocole de routage considéré). Les courbes sont calculées en partant des nœuds feuilles puis de proche en proche jusqu'au puits.

Pour la vérification, par *Model Checking*, que le nœud est capable de servir son trafic, nous utilisons les travaux de [Lampka et al., 2009], qui permettent de traduire des courbes *Network Calculus* en TA UPPAAL. Ces TA sont ensuite vérifiés avec un modèle du protocole étudié.

Nos contributions sont :

- l'ajout de la prise en compte des accès concurrents au médium aux travaux de [Schmitt and Roedig, 2005] et [Schmitt et al., 2007] ;
- l'application de la méthode de [Lampka et al., 2009] de traduction de courbes en TA pour vérifier des protocoles de RCsF. Nous augmentons aussi la portée

de cette technique en proposant une traduction en TA de courbes plus générales que celle de [Lampka et al., 2009].

Dans les sections suivantes, nous présentons le fonctionnement de *Network Calculus* et son application aux RCsF [Schmitt and Roedig, 2005] et [Schmitt et al., 2007]. Puis, nous présentons les courbes de *Network Calculus* qui peuvent être utilisées pour modéliser les RCsF ainsi que la technique de traduction de ces courbes en TA [Lampka et al., 2009]. Enfin, nous détaillons l'algorithme de vérification que nous proposons.

8.4.2 *Network Calculus pour les RCsF*

Nous présentons dans cette section, les fonctions et les opérateurs utilisés par le *Network Calculus* pour dériver les bornes de performances des réseaux étudiés. Nous nous intéressons au cas particulier des RCsF. Nous définissons les courbes de trafic, d'arrivées et de service et nous décrivons leur utilisation dans les RCsF. La plupart des définitions de cette section sont empruntées à [Le Boudec and Thiran, 2001], [Schmitt and Roedig, 2005] et [Wandeler, 2006].

Courbes de trafic cumulé

En *Network Calculus*, le trafic est représenté par des fonctions croissantes qui décrivent le trafic cumulé. Formellement, $R(t)$ est une fonction de $\mathbb{R}^+$ dans $\mathbb{N}$, telle que $R(a) \leq R(b)$ pour $a \leq b$ et $R(0) = 0$. Cette fonction représente le nombre d'événements qui arrivent entre l'instant 0 et l'instant t dans le système modélisé. Un événement correspond à l'arrivée d'un paquet ou d'un bit (dans ce document, nous considérons des paquets). Dans une définition alternative, appelée modèle fluide, $R(t)$ est définie de $\mathbb{R}^+$ dans $\mathbb{R}$, dans ce cas, un événement est considéré comme étant infinitésimal. La Figure 8.17 représente un exemple de $R(t)$ dans le cas où c'est une fonction de $\mathbb{R}^+$ dans $\mathbb{N}$.

On notera $R^*(t)$ la courbe en sortie du système étudié, cette courbe est définie de la même manière que $R(t)$ mais pour les événements qui sortent du système.

Le *backlog* du système est le nombre de paquets ou d'événements en cours de traitement dans le système. Il est donné par :

$$B(t) = R(t) - R^*(t) \tag{8.5}$$

Le retard virtuel du système est le retard subi par un événement au temps t, si tous les événements reçus avant t sont traités avant. Il est défini comme suit :

$$d(t) = \inf\{\tau \geq 0 | R(t) \leq R^*(t+\tau)\} \tag{8.6}$$

avec inf la fonction infinimum ¶.

¶. La notion d'infinimum est définie dans [Le Boudec and Thiran, 2001] : c'est la plus grande borne inférieure d'un sous-ensemble de $\mathbb{R}$ borné par le bas. Par exemple, les intervalles $[1, 3]$ et $]1, 3[$ ont le même infinimum : 1, dans le premier cas 1 est aussi le minimum car il est contenu dans l'intervalle, ce n'est pas vrai dans le second cas qui est un intervalle ouvert.

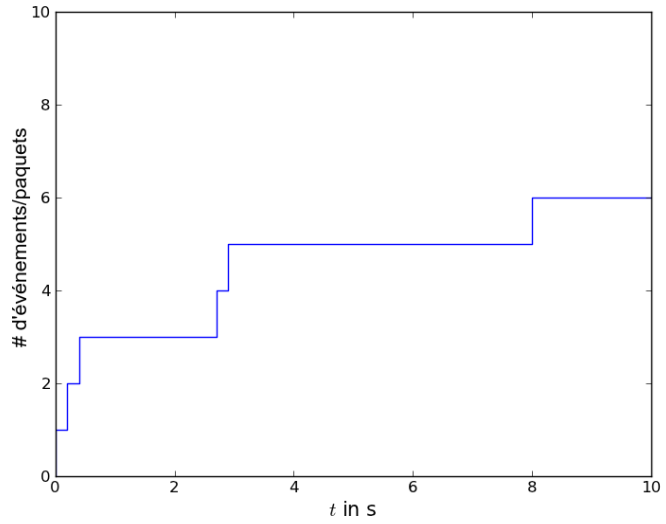


FIGURE 8.17 – Exemple de $R(t)$

On définit ensuite des bornes sur les courbes de trafic qui sont nommées courbes d'arrivées.

Courbes d'arrivées

En *Network Calculus*, les courbes d'arrivées sont utilisées pour exprimer des contraintes sur le trafic entrant ou sortant du système. Formellement, α est une courbe d'arrivées pour R si et seulement si pour tout $s \leq t$:

$$R(t) - R(s) \leq \alpha(t - s) \tag{8.7}$$

C'est-à-dire, α contraint le nombre d'événements qui peuvent arriver dans tout intervalle $[s, t]$ (par la suite on pose $\Delta = t - s$).

On peut de manière alternative (et équivalente) définir α de la manière suivante [Le Boudec and Thiran, 2001] :

$$R \leq R \otimes \alpha \tag{8.8}$$

avec $\otimes$ la convolution min-plus définie comme suit pour f et g deux fonctions croissantes :

$$(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t - s) + g(s)\} \tag{8.9}$$

La Figure 8.18 représente un exemple de courbe d'arrivées. Le comportement à court terme des arrivées R est borné par le début de la courbe α, d donne la distance minimum entre deux événements. Sur le long terme R est borné par des arrivées périodiques de période p. Cela signifie, par exemple que sur une période de 1 seconde 3 événements au maximum peuvent arriver, mais sur une période de 6 secondes seulement 5 événements au maximum peuvent arriver.

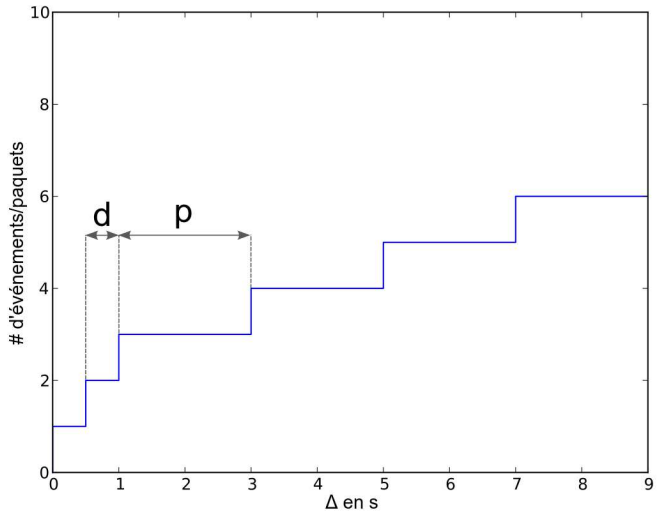


FIGURE 8.18 – Exemple de courbe d'arrivées $\alpha(\Delta)$

Courbes de service

Les courbes de services représentent le service minimum que peut fournir un nœud au trafic, c'est-à-dire le trafic qu'il peut écouler au minimum sur une période donnée. Dans la littérature [Le Boudec and Thiran, 2001], plusieurs définitions des courbes de service existent. Dans ce document, nous considérons les courbes de service simples, β, fonction croissante et $\beta(0) = 0$, est une courbe de service si et seulement si :

$$R^* \geq R \otimes \beta \tag{8.10}$$

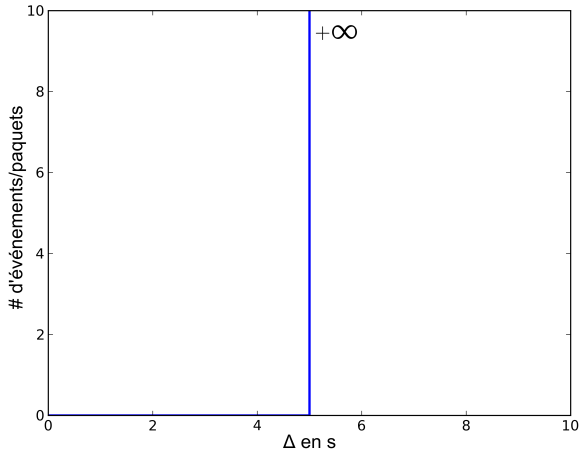


FIGURE 8.19 – Exemple de courbes de service $\beta(\Delta)$

La Figure 8.19 représente une courbe de service. Dans cet exemple, les événements sont servis au maximum après un délai de 5 secondes.

Real-Time Calculus

Il est à noter que [Wandeler, 2006] propose une version alternative de *Network Calculus* nommée *Real-Time Calculus* (RTC). Ses principales différences sont :

- Les courbes de trafic sont des fonctions à deux variables définies sur $\mathbb{R}$ et non plus sur $\mathbb{R}^+$.
- Les courbes d'arrivées et de services définissent des bornes supérieures et inférieures et non plus seulement supérieures pour les courbes d'arrivées et seulement inférieures pour les courbes de service.

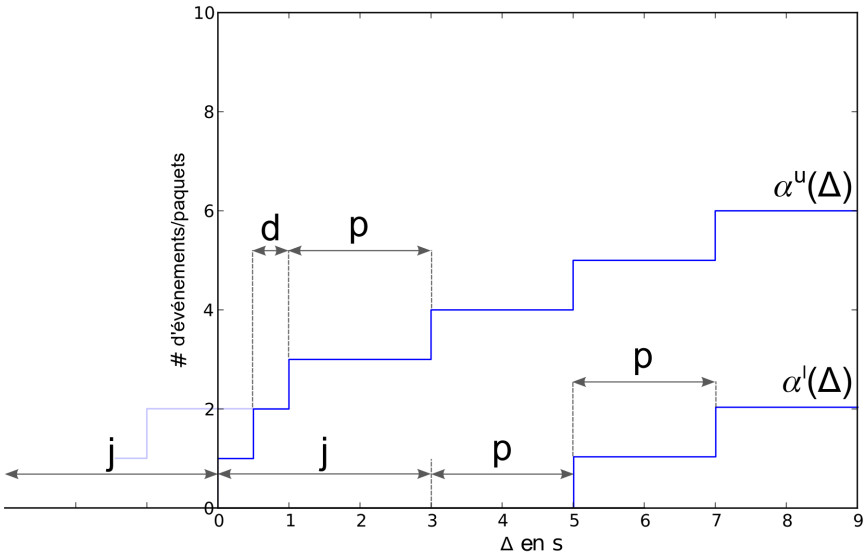


FIGURE 8.20 – Exemple de courbes d'arrivées RTC

La Figure 8.20 montre la représentation d'un modèle d'arrivées périodiques avec gigue (p, j, d) en *Real-Time Calculus*. Dans ce modèle, les événements ont une arrivée périodique p, une gigue j et une durée minimum entre deux événements d. A la différence de l'exemple 8.18 pour *Network Calculus*, ici un événement est forcé après un temps $j + p$ sans événement, ceci est représenté par la courbe α^l représentée sur la figure 8.20. Dans ce document, nous utilisons le *Network Calculus* car c'est le formalisme utilisé dans [Schmitt and Roedig, 2005] pour l'application aux RCsF. On note tout de même qu'une équivalence de *Network Calculus* et *Real-Time Calculus* est établie dans [Jouhet, 2012].

Sensor Network Calculus

Nous introduisons maintenant le *Sensor Network Calculus* qui est défini dans [Schmitt and Roedig, 2005]. C'est une application de *Network Calculus* pour les RCsF

qui permet de calculer les courbes d'arrivées sur le trafic d'entrée et de sortie des nœuds d'un RCsF. Nous définissons, dans cette section, toutes les formules nécessaires pour calculer ces courbes, qui représentent les interactions d'un nœud avec le reste du réseau. Cela nous permet de réaliser le premier point de notre méthode de vérification hybride : abstraire les interactions du nœud avec le réseau sous forme de courbes de *Network Calculus* (cf. section 8.4.1).

Nous commençons par donner la définition de la borne de *backlog* : elle contraint le nombre d'événements en cours de traitement dans le système. Elle n'est pas utilisée directement dans la méthode hybride que nous proposons, mais elle est utile pour dimensionner correctement la taille des files d'attente des nœuds du réseau :

$$B(t) \leq \sup_{s \geq 0}\{\alpha(s) - \beta(s)\} = v(\alpha, \beta) \tag{8.11}$$

La borne de retard contraint le délai subi par un événement dans un nœud :

$$d(t) \leq \sup_{s \geq 0}\{\inf\{\tau \geq 0 | \alpha(s) \leq \beta(s + \tau)\}\} = h(\alpha, \beta) \tag{8.12}$$

On remarque qu'en jouant sur la définition de β on fait varier cette borne : par la suite, nous choisissons une courbe de service qui permet de respecter la borne temporelle visée et nous vérifions que le protocole étudié est effectivement capable d'assurer ce service.

Dans [Schmitt and Roedig, 2005], la topologie du réseau est représentée sous forme d'arbre enraciné sur le puits, l'arbre dépend du protocole de routage considéré. Le trafic d'entrée d'un nœud i est la somme des trafics sortants de ses fils dans l'arbre et du trafic que lui-même génère :

$$\bar{R_i} = R_i + \sum_{j=1}^{n_i} R_j^* \tag{8.13}$$

avec R_i représentant les alarmes générées par le nœud i et R_j^* le trafic venant du fils j. Si on contraint ces trafics avec des courbes d'arrivées, on obtient l'équation suivante :

$$\bar{\alpha_i} = \alpha_i + \sum_{j=1}^{n_i} \alpha_j^* \tag{8.14}$$

avec α_i la courbe d'arrivées sur le trafic R_i et α_j^* sur R_j^*. α_i est une hypothèse sur le taux de détection d'alarmes de chaque nœud, on verra dans la section 8.4.3 quelles fonctions peuvent être utilisées. Les fonctions α^*, qui contraignent les courbes de trafic de sortie sont calculées en fonction de la courbe d'arrivées en entrée $\bar{\alpha}$ et de la courbe de service :

$$\alpha^* = \bar{\alpha} \oslash \beta \geq \bar{\alpha} \tag{8.15}$$

avec $\oslash$ la déconvolution min-plus définie comme suit pour f et g deux fonctions croissantes :

$$(f \oslash g)(t) = \sup_{s \geq 0}\{f(t + s) - g(s)\} \tag{8.16}$$

La courbe d'arrivées sur les sorties du nœud i est donc calculée comme suit :

$$\alpha_i^* = \bar{\alpha_i} \oslash \beta_i = (\alpha_i + \sum_{j=1}^{n_i} \alpha_j^*) \oslash \beta_i \tag{8.17}$$

Les courbes d'arrivées de chaque nœud peuvent être calculées de manière itérative, en partant des feuilles de l'arbre en remontant vers le puits grâces aux équations 8.14 et 8.17 comme représenté sur la Figure 8.21.

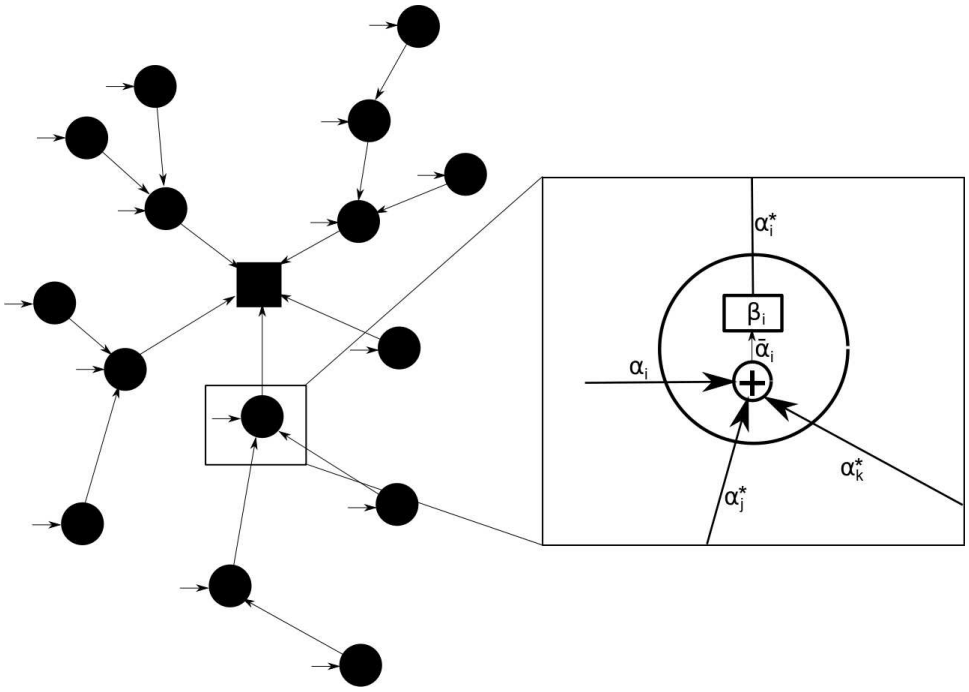


FIGURE 8.21 – *Sensor Network Calculus*

Un problème non abordé dans les travaux [Schmitt and Roedig, 2005] et [Schmitt et al., 2007] est le suivant : dans un réseau sans fil tel qu'un RCsF, des flux qui ne passent pas par le même nœud peuvent interférer. C'est dû à la nature *broadcast* du lien radio (présenté dans le Chapitre 2), un nœud subit donc, en plus des arrivées venant des fils de l'arbre, des interférences provenant de ses concurrents pour l'accès au canal et qui retardent ses propres accès. Pour les protocoles temps-réel, les accès doivent être priorisés, car si ce n'est pas le cas il y a une probabilité d'avoir une série de collisions qui produit un dépassement de l'échéance du paquet (cf. Chapitre 2). C'est d'ailleurs pour cela que nous proposons la coordonnée virtuelle du Chapitre 4. Le trafic compétiteurs prioritaires d'un nœud influe sur sa capacité à servir son trafic. On définit donc la notion de nœud compétiteur prioritaire :

Définition 8.4.1 *Les* ***compétiteurs prioritaires*** *d'un nœud n_a sont les nœuds interférants de n_a, qui ont du trafic à écouler et qui sont prioritaires par rapport au nœud n_a.*

On note que l'ensemble des nœuds compétiteurs prioritaires d'un nœud est défini par rapport au modèle d'interférence considéré, par exemple dans le cas d'un modèle d'interférences à deux sauts, ce sont les 2-voisins du nœud qui sont plus prioritaire que lui. Le service que peut fournir un nœud dans un réseau sans fil dépend donc du trafic de ses compétiteurs prioritaires. Pour prendre en compte cet aspect, nous définissons des courbes d'arrivées sur les trafics des nœuds prioritaires qui sont en compétition avec un nœud donné. Le détail du calcul de ces courbes d'arrivées est donné dans la section 8.4.4.

Dans la section suivante, nous nous intéressons aux courbes d'arrivées qui permettent de modéliser les bornes de trafic dans les RCsF et à leur traduction en TA UPPAAL.

8.4.3 Courbes de trafic et automates temporisés

Pour pouvoir vérifier, par *Model Checking*, que chaque nœud du réseau est capable de servir le trafic qui lui parvient, un modèle TA du comportement du nœud (le protocole de communication) et un modèle TA du trafic qui correspond à la courbe d'arrivées calculée pour ce nœud à l'aide de l'Equation 8.14 sont nécessaires. Dans cette section nous nous penchons sur les formes des courbes utilisées pour la modélisation du trafic dans les RCsF et à leur traduction en TA.

Intéressons nous d'abord à la forme des courbes d'arrivées que nous rencontrons pour borner la production de données dans les RCsF, c'est-à-dire la borne supérieure sur les paquets produits par un nœud suite à la détection d'un événement dans le réseau (noté α_i dans la section précédente). Pour modéliser ces bornes sur la production de paquets, nous utilisons une courbe qui représente un maximum de la fréquence de détection d'événement, comme dans [Schmitt and Roedig, 2005]. C'est-à-dire qu'une alarme va être produite, au plus, toute les δ secondes : concrètement, cela signifie que l'apparition des événements captés par un nœud est bornée par une fonction périodique. Tous les trafics qui sont en dessous de cette limite peuvent exister, notamment des apparitions apériodiques (tant que le nombre d'événements par unité de temps ne dépasse pas $\frac{1}{\delta}$). Il n'y a pas de borne inférieure : il est possible qu'il n'y ait pas d'événements. On considère qu'une alarme correspond à un paquet et que l'unité des courbes d'arrivées est le paquet (on se place dans le cas du modèle de trafic discret). Dans ce cas, la courbe d'arrivées est une fonction escalier de cette forme :

$$\gamma(\Delta) = \lceil \frac{\Delta}{\delta} \rceil \tag{8.18}$$

Pour un nœud feuille de l'arbre de routage, le trafic entrant vient seulement des mesures du capteur. La courbe d'arrivées sur le trafic que doit servir un nœud feuille $\bar{\alpha}$ est donc une fonction escalier. Si on fait l'hypothèse que les paquets sont servis avec un délai maximum de D_δ, la courbe de service est similaire à celle de la Figure 8.19. C'est le cas représenté par la Figure 8.22. La courbe d'arrivées sur le trafic de sortie du nœud feuille est alors définie par l'Equation 8.19.

$$\alpha^* = \alpha \oslash \beta = \lceil \frac{\Delta + D_\beta}{\delta} \rceil \leq \lceil \frac{\Delta}{\delta} \rceil + \lceil \frac{D_\beta}{\delta} \rceil \tag{8.19}$$

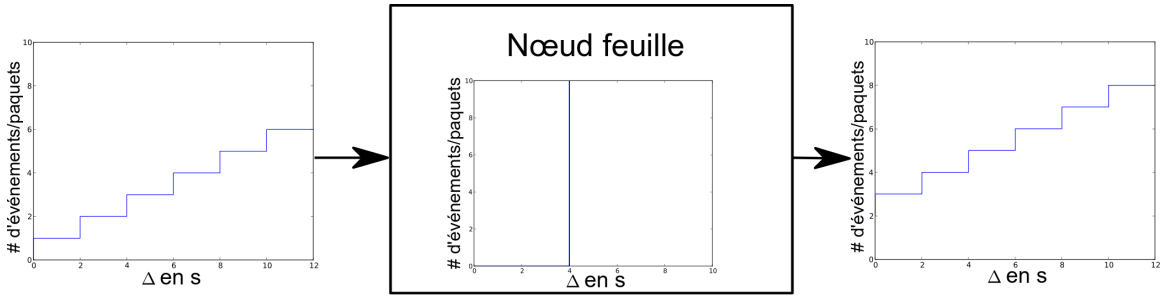


FIGURE 8.22 – Courbes d'arrivées sur l'entrée, courbe de service et courbe d'arrivées sur la sortie d'un nœud feuille

Pour obtenir un modèle TA de ce type de courbes d'arrivées nous utilisons la technique développée dans [Lampka et al., 2009]. Les auteurs proposent une technique pour représenter les courbes de d'arrivées de la forme suivante :

$$\gamma_N(\Delta) = N + \lceil \frac{\Delta}{\delta} \rceil \tag{8.20}$$

avec N un entier, représentant la possibilité d'arrivées d'événements simultanés. La technique de [Lampka et al., 2009] est donc applicable à notre problème en posant $N = \lceil \frac{D_\beta}{\delta} \rceil$. On peut noter que pour $N = 0$, on retombe dans le cas de l'Equation 8.18. La Figure 8.23 représente un TA capable de générer toutes les courbes qui respectent la courbe d'arrivées définie par l'Equation 8.20. Le double cercle représente l'état initial, dans cet état, la transition du haut est forcée au bout d'un temps égal à `delta` (qui correspond au δ de l'Equation 8.20). Cette transition incrémente la variable b si elle est inférieure à sa valeur maximale `N` (qui correspond au N de l'Equation 8.20). Depuis l'état initial, il est aussi possible de prendre la transition vers l'état *Committed* (C) si `b` est supérieur à 0. Cette transition envoie un signal sur la variable canal *broadcast event*. Ensuite, l'horloge x est remise à zéro si `b` == `N` et `b` est décrémenté. Un événement peut donc être envoyé seulement si `b` est supérieur à 0, `b` est incrémenté tout les `delta` par la transition du haut. Il est donc possible d'avoir `N` arrivées d'événements sans contraintes d'espacement temporel et ensuite l'événement suivant devra être séparé d'au moins `delta` secondes. Ce comportement respecte bien la courbe d'arrivées définie par l'Equation 8.20, comme prouvé dans [Lampka et al., 2009].

On se place maintenant dans le cas général et plus seulement dans celui des nœuds feuilles. Comme représenté sur la Figure 8.21 et défini par l'Equation 8.17, des courbes d'arrivées doivent être sommées. On va donc s'intéresser à la somme de fonctions escaliers. Il est possible qu'on ait à sommer des fonctions qui ont des périodes δ différentes, notamment si les périodes de mesure de différents capteurs ne sont pas

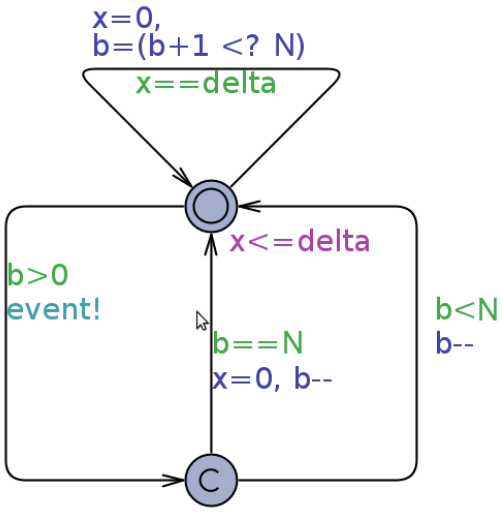


FIGURE 8.23 – Automate capable de générer des courbes contraintes par la courbe d'arrivées définie par l'Equation 8.20

égales. La somme de deux fonctions escaliers δ_a et δ_b est donc :

$$\gamma_{sum} = N_a + \lceil \frac{\Delta}{\delta_a} \rceil + N_b + \lceil \frac{\Delta}{\delta_b} \rceil \tag{8.21}$$

γ_{sum} est aussi une fonction escalier. Jusqu'ici les courbes escaliers que nous obtenions avaient des "marches" régulières de largeur δ. En revanche, comme on peut le constater sur la Figure 8.24, γ_{sum} n'a pas obligatoirement de marches régulières.

On constate tout de même que la somme des deux courbes possède un motif périodique. En $\Delta = 0$, les marches des deux courbes escaliers sont "synchronisées". Ensuite, pour $0 \leq \Delta < p$ la somme forme un motif de marches irrégulières. Ce motif va se répéter à partir du moment où les deux courbes sommées se "resynchronisent". La somme des deux courbes est donc une courbe dont le motif de base se répète avec une période p, p correspondant au temps entre deux "synchronisations" :

$$p = x_m.\delta_a = y_m.\delta_b \text{ avec } \{x_m, y_m\} = \min\{x \geq 1, y \geq 1 | x.\delta_a = y.\delta_b\} \tag{8.22}$$

Si δ_a et δ_b sont des entiers, p correspond alors au plus petit commun multiple (ppcm) de δ_a et δ_b :

$$p = \text{ppcm}(\delta_a, \delta_b) = \frac{\delta_a.\delta_b}{\text{pgcd}(\delta_a, \delta_b)} \tag{8.23}$$

Cependant, δ_a et δ_b peuvent être des réels dans le cas où la période minimale de production des alarmes n'est pas entière (par exemple, on peut avoir un système qui produit au maximum 1 alarme toutes les 3,68 secondes). Dans ce cas, pour implémenter le calcul, on considère que ces valeurs ont une précision finie notée pr, c'est-à-dire que δ a pr décimales. On peut alors remplacer δ_a et δ_b respectivement par $\delta_a.10^{pr}$ et

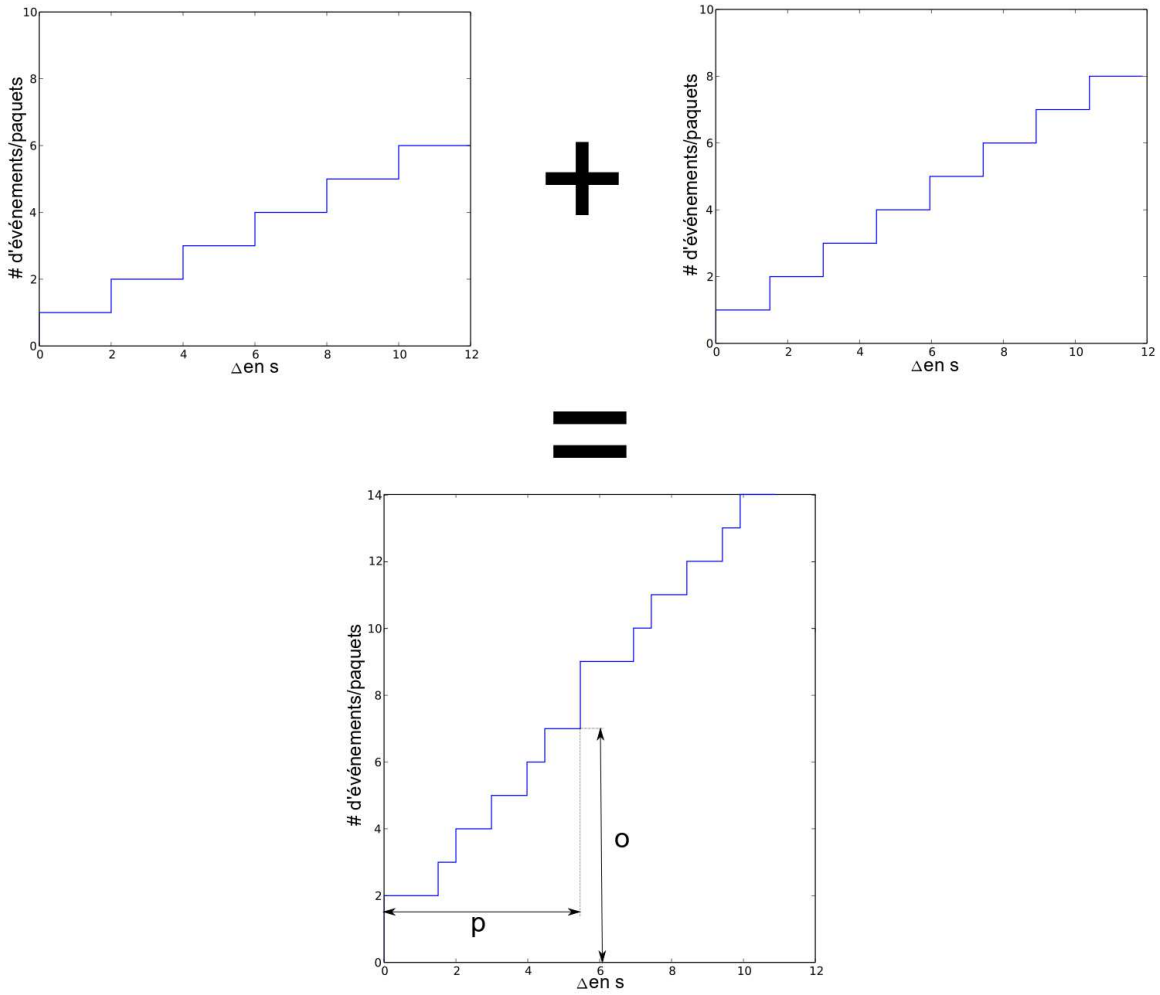


FIGURE 8.24 – Somme de deux courbes escaliers

$\delta_b.10^{pr}$, on obtiendra p en divisant le résultat de l'Equation 8.23 par 10^{pr}. Avoir une précision finie dans les bornes sur les arrivées n'est pas un problème en réalité, car le temps dans systèmes étudiés est discret et donc une précision plus importante dans la borne que le temps d'un cycle processeur ou celui de l'émission d'un bit est inutile.

[Lampka et al., 2009] ne fournit pas de TA pour ce type de courbes, nous nous intéressons donc à la représentation de ce type de courbes escaliers sous forme de TA pour pouvoir les intégrer dans le processus de *Model Checking*. Nous commençons par prouver que le respect de la première période de la courbe escalier ($\Delta \leq p$) suffit à respecter toute la courbe, nous proposons ensuite un TA qui produit des événements qui respectent la première partie d'une courbe escalier périodique.

Théorème 8.4.1 *Si $\alpha(\Delta)$, une courbe escalier périodique de période p et $R(t) - R(s) \leq \alpha(t-s)$ $\forall \Delta = t - s \leq p$ alors $\alpha(\Delta)$ est une courbe de service pour le flux R.*

Preuve On suppose que l'on a $R(t) - R(s) \leq \alpha(t-s)\ \forall \Delta = t - s \leq p$, par définition on a aussi $R(kt) - R(ks) \leq k\alpha(t-s)$ avec $k \in \mathbb{N}$ (si $\alpha(\Delta)$ événements peuvent arriver en Δ secondes, au maximum $k\alpha(\Delta)$ événements peuvent arriver en $k\Delta$ secondes avec $k\Delta = kt - ks$) donc on doit prouver que $\forall \Delta' \in \mathbb{R}$, $k\alpha(\Delta) \leq \alpha(\Delta')$. Or $\alpha(\Delta')$ est une courbe escalier périodique de période p, on a donc $\alpha(\Delta') = \alpha(\Delta + k.p) = \alpha(\Delta) + k.\alpha(p)$ $\forall \Delta \leq p$ et donc :

$$R(kt) - R(ks) \leq k\alpha(\Delta) \leq k\alpha(p) \leq \alpha(\Delta')$$

∎

Le théorème 8.4.1 signifie qu'un TA qui produit des événements qui respectent la première période d'une courbe de service, produit en fait des événements qui respectent toute la courbe de service escalier périodique. La figure 8.25 représente un automate capable de générer des événements qui respecte une courbe d'arrivées escalier périodique où N est la hauteur de la première marche de la période et M le nombre de marches de la période. Quand la transition est sélectionnée, un événement est envoyé sur le canal *event* et l'horloge qui correspond à l'événement courant x_i est remise à zéro. Pour pouvoir sélectionner la transition et donc envoyer l'événement, il faut que les événements précédents se soient produits il y a suffisamment longtemps pour que la courbe d'arrivées soit respectée : il faut que le Nième événement (N est la taille de la première marche) avant l'événement courant se soit produit à une distance temporelle au moins égale à la longueur de la première marche de la courbe escalier (noté Δ_N). Il faut que $(N+1)$ième événement avant le courant soit à plus de la longueur des deux premières marches (noté Δ_{N+1}) et ainsi de suite pour toutes les marches correspondant à une période.

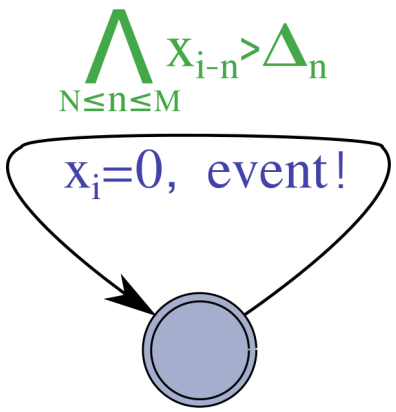


FIGURE 8.25 – Automate capable de générer des courbes contraintes par une courbe d'arrivées escalier périodique

L'automate de la Figure 8.25 permet de respecter la première période de la courbe de service escalier périodique. Grâce au théorème 8.4.1 on est sûr qu'il produit uniquement des séries d'événements qui respectent la courbe d'arrivées encodée par les Δ_n. Cependant, cet automate a un inconvénient : M horloges sont néces-

saires pour garder en mémoire les dates des M événements précédents, or la complexité du *Model Checking* augmente exponentiellement avec le nombre d'horloges [Baier and Katoen, 2008]. Pour éviter ce problème nous utilisons [Künzli et al., 2007], où les auteurs montrent que les courbes d'arrivées de *Real-Time Calculus* peuvent être approximées de manière conservative par un modèle d'arrivées périodiques avec gigue (p, j, d) déjà évoqué dans la section 8.4.2 et illustré par la Figure 8.20 (notons que comme nous travaillons avec *Network Calculus* nous utilisons seulement la courbe d'arrivées correspondant à la borne supérieure). Une méthode pour approximer de manière conservative les courbes escaliers périodiques sous forme de l'Equation 8.20 (p.188) est fournie, la fonction qui approxime γ_{sum} (Equation 8.21 p.189) est :

$$\gamma_a = o + \lceil \frac{\Delta}{p} \rceil \tag{8.24}$$

avec p défini par l'Equation 8.23 et $o = \gamma_{sum}(p)$. Nous pouvons donc utiliser l'automate 8.23 qui ne nécessite qu'une seule horloge. Cependant comme on majore la courbe d'arrivées, cela donne une borne sur le délai plus "lâche" et cela augmente le nombre de cas générés par le *model checker*.

8.4.4 Algorithme de vérification

L'algorithme de vérification que nous proposons permet de calculer, pour chaque nœud, les courbes d'arrivées en entrée et en sortie du nœud, et vérifier par *Model Checking* si le nœud a un comportement correct quand il est soumis à ce trafic. Lors du modèle checking, les TA qui représentent les courbes d'arrivées génèrent de manière exhaustive tous les trafics qui respectent ces courbes d'arrivées. Pour calculer les courbes d'arrivées nous utilisons les outils décrits dans le *Sensor Network Calculus* [Schmitt and Roedig, 2005]. Pour les courbes de service des nœuds nous faisons l'hypothèse qu'un nœud est capable de servir tout le trafic qu'il reçoit avec un délai maximum imposé par le protocole utilisé (on utilise donc une courbe de la forme de celle de la Figure 8.19), c'est cette hypothèse qui est vérifiée lors du *Model Checking*. Nous ajoutons aussi la prise en compte de la compétition dans l'accès au médium.

La compétition dans l'accès au médium fait qu'un nœud peut y accéder plus ou moins rapidement en fonction de la taille de son domaine d'interférences et des priorités des nœuds interférants. La prise en compte des compétiteurs d'un nœud n_a consiste à calculer pour n_a, une courbe d'arrivées des événements sur le canal causés par les compétiteurs qui sont plus prioritaires que n_a pour l'accès au médium. Cette courbe est la somme des courbes d'arrivées sur les sorties des nœuds compétiteurs plus prioritaires (ensemble noté $N_{cp}^{n_a}$) :

$$\alpha_{cp(i)} = \sum_{i \in N_{cp}^{n_a}} \alpha_i^* \tag{8.25}$$

La procédure de vérification consiste à utiliser un outil de *Model Checking* pour vérifier que chaque nœud du réseau respecte bien la courbe de service donnée en hypo-

thèse. Et ce, sous tous les régimes de trafic d'entrée et tous les régimes d'interférences dues aux compétiteurs, qui sont bornés respectivement par $\bar{\alpha_i}$ et $\alpha_{int(i)}$ pour un nœud n_i.

Algorithm 8.4.1 Algorithme de vérification

Require: arbre de routage T, hypothèses de trafic α_i, hypothèse de service β, ensemble de propriétés P, modèle UPPAAL du protocole étudié TA_p.

1: *Nœuds*=trieNœuds(T) ▷ *Nœuds* est l'ensemble des nœuds triés par rang dans l'arbre et priorité
2: **for** $n_i \in Nœuds$ **do**
3: $\bar{\alpha_i} = \alpha_i$
4: **for** $n_j \in N_c^i$ **do** ▷ N_c^i est l'ensemble des fils du nœud n_i
5: $\bar{\alpha_i} = \bar{\alpha_i} + \alpha_j^*$
6: **end for**
7: $\alpha_i^* = \bar{\alpha_i} \oslash \beta_i$
8: **for** $n_k \in N_{cp}^i$ **do** ▷ N_{cp}^i est l'ensemble des compétiteurs prioritaires du nœud n_i
9: $\alpha_{cp(i)} = \alpha_{cp(i)} + \alpha_k^*$
10: **end for**
11: TA_{in}=produceAutomaton($\bar{\alpha_i}$)
12: TA_{cp}=produceAutomaton($\alpha_{cp(i)}$)
13: NTA=produceModel(TA_p,TA_{in},TA_{cp})
14: result=UPPAAL-check(P,NTA)
15: **if** result==0 **then**
16: Erreur détectée
17: **return** Traces
18: **end if**
19: **end for**

La procédure de vérification est détaillée par l'Algorithme 8.4.1. Il prend en entrée, une représentation du réseau sous forme d'arbre enraciné au puits. À chaque nœud n_i peut être attaché une courbe d'arrivées sur les événements mesurés α_i. On donne une courbe de service dont le délai D dépend du protocole utilisé (on prend le délai maximum théorique du protocole). L'algorithme prend aussi en entrée un ensemble de propriétés P que chaque nœud doit vérifier, cet ensemble devra comporter la propriété "tous les paquets sont servis dans le délai D". Il peut aussi comporter toute autre propriété qui permet vérifier le bon fonctionnement du protocole (absence d'état bloquant, etc). Les nœuds sont classés par ordre de nombre de sauts pour atteindre le puits et priorité (ligne 1), de cette manière, les calculs sont faits d'abord pour les nœuds feuilles (par ordre de priorité d'accès au canal) et ensuite de proche en proche vers le puits. Les lignes 3 à 7 de l'Algorithme 8.4.1 sont une implémentation de l' Equation 8.17 qui permet de calculer la courbe de sortie d'un nœud à partir des courbes de sorties des fils du nœud, de la courbe d'arrivées sur les mesures et de la courbe de service. Les lignes 8 à 10 implémentent l'Equation 8.25. Les lignes 11 et 12 permettent de créer des TA qui permettent de mimer le comportement des

fils et des nœuds compétiteurs, les arrivées d'événements de ces TA sont produites par l'automate présenté dans la Figure 8.23. Ces deux automates simples modélisent toutes les interactions du réseau avec le nœud. Les lignes 12 et 13 permettent, pour le nœud courant, de produire le modèle qui est vérifié avec le *model checker* UPPAAL. Enfin, les lignes 14 à 16 permettent de stopper la vérification à la volée si un nœud viole une des propriétés vérifiées.

Cette méthode de vérification permet d'abstraire tout le réseau par deux automates et de vérifier que l'interaction entre un automate qui représente le comportement du nœud et ces deux automates est correcte. La vérification pour un nœud nécessite donc une durée raisonnable (car le réseau est abstrait). La durée de vérification du réseau augmente donc linéairement avec le nombre de nœuds à vérifier. Dans le cas où l'on ne fait pas l'abstraction (*Model Checking* pur) le temps de vérification augmente exponentiellement avec le nombre de nœuds dans le réseau (car chaque nœud augmente la taille d'un état et le nombre d'horloges comme on a pu le constater dans la section 8.2). Cette méthode contribue donc grandement au passage à l'échelle dans la vérification exhaustive des protocoles temps-réel pour RCsF comme on le vérifiera dans le Chapitre 9.

On note que la durée de vérification d'un nœud dépend de la forme de la courbe de service : si la courbe est "haute" (contrainte "lâche") de nombreux profils de trafic doivent être explorés lors du *Model Checking*. Il est possible d'améliorer les performances de cette technique de vérification en vérifiant seulement un sous-ensemble des nœuds du réseau. On utilise pour cela le constat que la vérification se fait sur tous les profils de trafic rendus possibles par les courbes d'arrivées associées au nœud : cela signifie que si un nœud a des courbes d'arrivées plus hautes que tous les autres nœuds du réseau (N maximal et δ minimal : la courbe d'arrivées du nœud est strictement supérieure aux autres) et qu'il respecte les propriétés alors tous les autres nœuds les respectent aussi car les profils de trafic auxquels ils peuvent être soumis ont déjà été vérifiés lors du *Model Checking* du premier nœud.

L'application au protocole RTXP de la méthode hybride présentée dans ce chapitre et l'évaluation de ses performances sont présentées dans le Chapitre 9.

8.4.5 Conclusion sur la méthode hybride de passage à l'échelle

Comme nous l'avons constaté dans les sections précédentes de ce chapitre, le principal problème du *Model Checking* est son incapacité à vérifier des réseaux de grandes tailles. C'est un problème pour les RCsF qui peuvent compter jusqu'à des centaines de nœuds. Pour atténuer ce problème nous proposons une méthode hybride qui allie le *Network Calculus* qui permet de passer l'échelle et le *Model Checking* qui permet d'avoir une vérification exhaustive des comportements du protocole étudié. Cette méthode utilise et étend les travaux de [Schmitt and Roedig, 2005] sur le *Network Calculus* adapté eu RCsF et de [Lampka et al., 2009] qui traduisent les courbes en TA.

Notre méthode consiste à, pour chaque nœud du réseau, abstraire les interactions avec le réseau par des courbes d'arrivées et vérifier par *Model Checking* que le nœud est capable de servir les arrivées en temps borné. Le modèle comporte seulement trois TA et est donc vérifiable par *Model Checking* (il faut tout de même que le modèle du protocole ne définisse pas un nombre trop important d'horloges qui induise une augmentation exponentielle de la durée de la vérification). Dans le Chapitre 9 nous appliquons cette méthode à RTXP, cela nous permet de constater qu'il est possible de passer l'échelle avec cette méthode lors de l'étude d'un protocole réel et de vérifier formellement le bon fonctionnement de RTXP. Cette technique de vérification s'inscrit dans une méthodologie de vérification de protocoles temps-réel pour RCsF présentée dans la section suivante.

8.5 Méthodologie globale de vérification formelle des RCsF

Dans cette section, nous résumons le déroulement de la méthodologie de vérification proposée au travers des trois contributions de ce chapitre. Nous donnons ensuite un exemple simple de l'application de cette méthodologie.

8.5.1 Présentation de la méthodologie

La méthodologie méthodologie proposée est illustrée par la Figure 8.26, elle est centrée autour de la modélisation du protocole étudié : le but de la méthodologie est de garantir le bon fonctionnement d'un modèle du système pour pouvoir ensuite l'implémenter. Il faut donc produire en premier lieu un modèle formel sous forme de TA qui est issu des spécifications du protocole, ce modèle doit éviter l'explosion combinatoire de l'espace de configurations et doit donc utiliser la modélisation des transmissions *broadcast* préconisée lors de l'étude de la section 8.2. Ensuite, nous proposons une méthodologie en trois étapes de vérification :

- La première étape consiste à effectuer une vérification seulement temporelle de ce modèle pour détecter les premières erreurs de conception et modélisation, si des erreurs sont détectées, elle doivent être corrigées et le modèle vérifié de nouveau jusqu'à ce que plus aucune erreur ne soit détectée. Cette première étape de vérification s'effectue sur des topologies simples du fait du problème d'explosion combinatoire.
- La deuxième étape est la vérification avec prise en compte des liens non-fiables présenté dans la section 8.3. Cette méthode permet de détecter des erreurs dues à la dynamique de la topologie et de quantifier leurs probabilités d'apparition. Cela permet une nouvelle fois de modifier le modèle pour permettre d'éviter les erreurs.
- La troisième étape, est la vérification du protocole à l'aide de la méthode hybride *Network Calculus* et *Model Checking* présentée dans la section 8.4. Elle permet de détecter et corriger les erreurs qui apparaissent seulement dans des réseaux de grandes tailles.

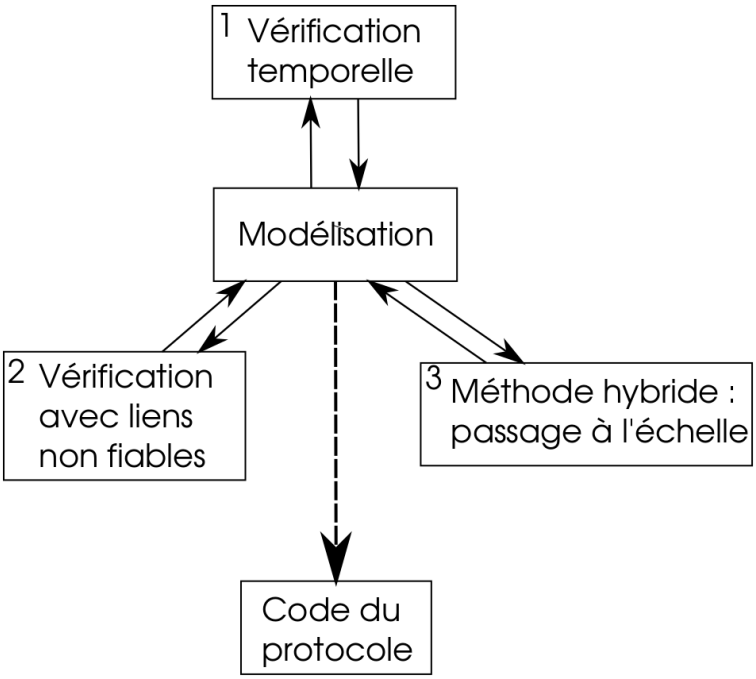


FIGURE 8.26 – Récapitulatif de la méthodologie de vérification formelle

Le code final du protocole qui sera embarqué sur les capteurs doit être produit à partir du modèle corrigé au cours des trois étapes de vérification. Cette méthodologie est complémentaire des simulations et implémentations sur *testbed*.

8.5.2 Exemple illustratif d'application de la méthodologie de vérification

Pour illustrer brièvement l'utilisation de cette méthodologie, avant de l'appliquer à RTXP dans le Chapitre 9, nous considérons un réseau de trois nœuds tous à portée radio (c'est une clique) représenté par le Figure 8.27. Nous considérons que le protocole vérifié est un protocole d'allocation de *slot* TDMA : au début de chaque trame, les nœuds participent à un tournoi pour obtenir des *slots*, le détail du protocole n'est pas donné ici, le but de cet exemple n'est pas de vérifier un protocole réel mais d'illustrer notre méthodologie de vérification. On veut vérifier que les nœuds A, B et C obtiennent chacun un *slot* en temps borné et peuvent donc transmettre des paquets sans collisions.

Avant la vérification en elle-même, il y a une étape de modélisation du protocole étudié. Durant cette étape, l'utilisateur produit un modèle UPPAAL du protocole étudié en utilisant la méthode de modélisation des transmissions *broadcasts* préconisée par l'étude de la section 8.2. On rappelle que la topologie est modélisée par une matrice d'adjacence, dans le cas du réseau de la Figure 8.27, la matrice ne contient que des

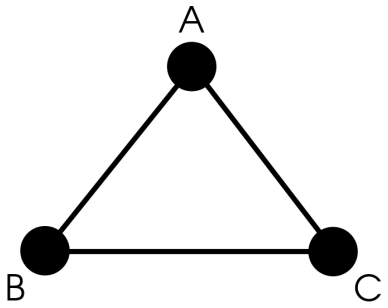


FIGURE 8.27 – Topologie utilisée pour illustrer l'utilisation de la méthodologie de modélisation proposée.

1 :

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{8.26}$$

Nous détaillons le processus de modélisation lors de l'application de la méthodologie de vérification à RTXP dans le Chapitre 9.

La première étape de vérification se fait directement dans UPPAAL avec le modèle du protocole incluant la topologie sous forme de matrice d'adjacence. Dans ce cas on vérifie que les nœuds A, B et C obtiennent un *slot* différent (sinon il y a collision) en temps borné, dans le cas du réseau clique. Si le modèle ne respecte pas cette propriété, UPPAAL fournit des contre-exemples qui permettent de déceler la source d'erreur et de corriger le fonctionnement du protocole. Une fois assuré que le protocole est capable d'allouer les *slots* lorsque les échanges de paquets sont sûrs, nous passons à la deuxième étape de vérification : la méthode présentée dans la section 8.3. Dans ce cas, des probabilités de succès de transmissions sont attribuées aux liens, dans notre exemple nous considérons que la probabilité de réception sur les trois liens de la Figure 8.27 est de 0.9. Le protocole va être vérifié sur les topologies représentées par la Figure 8.28 (la topologie (1) étant la topologie de base). Imaginons que le protocole,

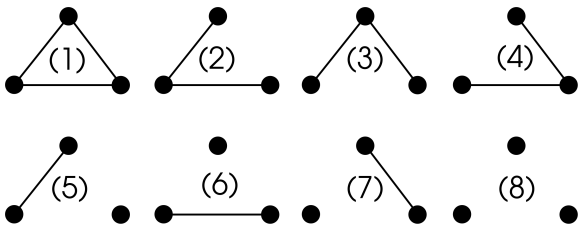


FIGURE 8.28 – Topologies vérifiées dans la deuxième étape : prise en compte des liens non-fiables.

en l'état, n'est pas capable de résoudre le problème du terminal caché présenté page 15 dans le Chapitre 2. Ce problème apparaît dans les topologies (2), (3) et (4), UPPAAL rend donc une réponse négative pour ces topologies. Notre méthode permet, en plus, d'évaluer la probabilité d'apparition du problème : d'après l'Equation 8.2

et l'Algorithme 8.3.2 la probabilité d'être dans un cas de terminal caché est de 0.24 ($0.9 \times 0.1 \times 0.9 + 0.9 \times 0.9 \times 0.1 + 0.1 \times 0.9 \times 0.9$). En fonction de la valeur de cette probabilité, le concepteur du protocole peut décider ou non de modifier le fonctionnement du protocole pour prendre en compte ce cas (typiquement si la valeur de la probabilité est très petite, prendre en compte le cas dans le protocole peut contraindre à surdimensionner fortement le système). Ici, dans près d'un quart des cas, il y aura une erreur, le protocole doit donc être corrigé. Ces deux premières étapes de vérification se font sur des réseaux de petites tailles. Cependant, les RCsF sont des réseaux pouvant contenir des centaines de nœuds avec des voisinages de dizaines de nœuds, la troisième étape de vérification permet de vérifier que les protocoles passent l'échelle sans introduire de mauvais fonctionnements. Cette étape peut notamment être utilisée pour vérifier le fonctionnement du protocole sur une topologie qui correspond au déploiement réel du réseau. L'algorithme 8.4.1, qui implémente notre méthode, requiert les informations suivantes :

1. l'arbre de routage : nous considérons donc que nous avons à disposition la topologie logique du déploiement visé ainsi qu'un arbre de routage sur cette topologie.
2. une hypothèse de service sous forme de courbe de service : on prend comme hypothèse que le trafic est servi en temps borné, cela correspond à la courbe de la Figure 8.29, c'est cette hypothèse qui est vérifiée par *Model Checking*.

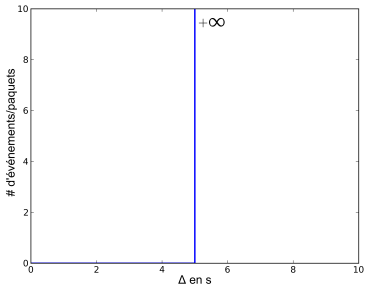


FIGURE 8.29 – Exemple de courbe de service (paquets servis au maximum après un délai de 5s)

3. une hypothèse sur le trafic sous forme de courbe d'arrivées : les courbes d'arrivées utilisées sont présentées dans la section 8.4.3 : dans le pire cas le trafic détecté par un nœud donné est périodique dans le cas le plus favorable il n'y a pas de détection d'événement. Un exemple de ce type de courbe est illustré par la Figure 8.30.
4. un ensemble de propriétés qui permettent de vérifier l'hypothèse sur la courbe de service : cela correspond, dans le cas de notre exemple, à vérifier que chaque nœud obtient un *slot* en temps borné.
5. un modèle UPPAAL du protocole issu de l'étape de modélisation.

Pour chaque nœud un ensemble de compétiteurs prioritaires est défini en fonction de la compétition pour l'allocation des *slots* : le protocole, pour être déterministe,

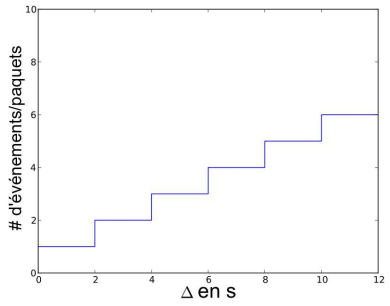


FIGURE 8.30 – Exemple de courbe d'arrivées d'alarmes

doit définir un ordre de priorité qui permet de définir un seul gagnant durant la phase de compétition. Cette information permet de déterminer pour chaque nœud i, quels nœuds vont obtenir des *slots* avant le nœud i et donc retarder l'écoulement de son trafic, cela va donc influer sur la capacité du nœud à écouler son trafic en temps borné. L'algorithme de vérification (Algorithme 8.4.1) permet de vérifier que sous les hypothèses de trafic faites, chaque nœud est capable de servir le trafic qui correspond à l'hypothèse de service.

Cette dernière étape de vérification permet de détecter les problèmes de conception du protocole qui apparaissent dans les réseaux larges échelles, mais aussi les problèmes dus au déploiement des nœuds : par exemple, les cas où un seul nœud doit relayer beaucoup de trafic par rapport aux autres et dépasse donc la capacité du protocole, ou bien un nœud qui possède de nombreux compétiteurs prioritaires et qui ne peut donc pas obtenir de *slots*.

Cette exemple simple nous permet d'illustrer concrètement l'utilisation de la méthodologie de vérification proposée, nous allons plus loin dans le Chapitre 9 en appliquant la méthodologie à RTXP.

8.6 Conclusion

Dans ce chapitre, nous proposons des techniques qui permettent la vérification formelle de protocoles temps-réel pour RCsF. Ces techniques répondent aux caractéristiques des RCsF : la nature *broadcast* et la non-fiabilité des liens sans fil et la grande taille de ces réseaux. Nous proposons trois contributions qui permettent de répondre à ces aspects :

- Une étude des différentes modélisations existantes dans la littérature des transmissions *broadcast*. Elle conclut que la méthode qui vérifie la connectivité avant la synchronisation permet de limiter le problème d'explosion combinatoire lors du *Model Checking* et doit donc être préférée aux autres pour la modélisation des protocoles de RCsF.
- Une technique de vérification qui prend en compte les liens non-fiables : cette méthode consiste à vérifier les topologies rendues possibles par l'apparition et la disparition transitoire des liens radios, elle est indépendante de l'outil de vérification utilisé.

- Une technique de vérification hybride *Network Calculus* et *Model Checking* qui permet de tirer partie de la capacité de *Network Calculus* d'abstraire les interactions du réseau avec un nœud et la capacité du *Model Checking* d'effectuer une exploration exhaustive des comportements possibles du protocole et qui garantit donc un haut niveau de confiance.

Ces trois contributions sont articulées pour donner forme à une méthodologie globale de vérification dans la section 8.5. Cette méthodologie est appliquée à RTXP dans le chapitre suivant, cela permet à la fois de vérifier formellement RTXP et d'évaluer l'applicabilité de la méthodologie proposée.

Chapitre 9

Application de la méthodologie de vérification à RTXP

Dans le Chapitre 5 nous proposons RTXP, un protocole *cross-layer* (MAC et routage), localisé, adapté au trafic et temps-réel pour RCsF. Nous évaluons ses performances par simulation et les comparons positivement à des solutions de la littérature. Cependant, la simulation n'est pas suffisante pour garantir le respect des spécifications, car cette technique n'explore qu'une partie des comportements du protocole. Les applications critiques nécessitent un niveau de confiance plus élevé qui est fourni par les méthodes de vérification formelle.

Dans ce chapitre, nous appliquons la méthodologie de vérification pour RCsF proposée dans le Chapitre 8 à RTXP. Cela nous permet à la fois de nous assurer que RTXP a un fonctionnement correct (respect des échéances) et d'étudier l'applicabilité de la méthodologie de vérification proposée. Nous détaillons d'abord la modélisation en TA UPPAAL de RTXP qui s'appuie sur l'étude de la section 8.2 du Chapitre 8, nous détaillons en particulier la modélisation des transmissions, des collisions et des comportements temporels (périodes de *backoff* par exemple). A partir de cette modélisation nous vérifions le comportement de RTXP, d'abord avec des liens fiables (vérification temporelle seulement), car cela permet de vérifier que le protocole fonctionne correctement dans un cas idéal. Puis, pour prendre en compte la probabilité de pertes de paquets, nous appliquons la méthode présentée dans la section 8.3 du Chapitre 8. Nous utilisons ensuite la méthode qui allie *Network Calculus* et *Model Checking* (section 8.4 Chapitre 8) pour vérifier des réseaux plus importants.

Les résultats de vérification de RTXP nous permettent de conclure que le protocole permet effectivement de garantir la livraison de paquets en temps borné dans les RCsF.

9.1 Modélisation de RTXP

Pour modéliser RTXP, nous utilisons l'outil UPPAAL. En effet, comme mentionné dans la section 7.3 du Chapitre 7, c'est l'outil qui permet le mieux de modéliser tous les aspects des RCsF. Notamment, la possibilité de définir des tableaux de données

et de réaliser leur mise à jour avec des fonctions dans un langage proche du C rend cet outil très puissant. Dans cette section, nous détaillons la modélisation du protocole RTXP dont le fonctionnement est détaillé dans la section 5.5 du Chapitre 5. le fonctionnement de RTXP possède deux aspects principaux à modéliser :

1. le *backoff* qui consiste à attendre pour une durée prédéterminée en scrutant le canal ;
2. les transmissions (et collisions) de paquets de données et de *jamming codes*.

9.1.1 Modélisation du *backoff*

On rappelle que RTXP est composé de 4 phases temporelles (cf. Figure 5.1 p.5.1 du Chapitre 5) : une phase de *backoff* qui permet de réserver l'accès au canal de manière déterministe, une phase de transmission du paquet de données, une phase de *backoff* qui permet de sélectionner le nœud relayeur et un *slot L* qui permet de répéter les trois premières phases pour les nœuds qui n'ont pas eu accès au canal et qui ont un paquet à transmettre.

Les phases de *backoff* consistent à scruter le canal pour une durée déterminée (dans le cas de RTXP la durée est une fonction de la coordonnée du nœud, cf. section 5.5 du chapitre 5). Si un *jamming code* est détecté avant l'écoulement de la durée, le nœud perd la contention, sinon il émet un *jamming code* pour signifier qu'il est le gagnant de la contention.

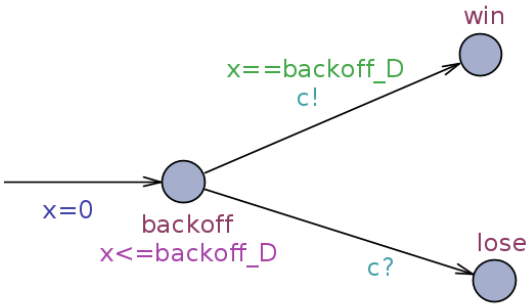


FIGURE 9.1 – Modélisation du backoff en TA UPPAAL

La Figure 9.1 représente la modélisation en TA UPPAAL du *backoff*. La transition qui permet de passer dans l'état *backoff* remet l'horloge x à zéro, cette transition a pour origine un état qui correspond à un comportement du nœud qui précède le *backoff*. L'état *backoff* correspond à l'état d'attente et de scrutation du canal, le temps peut s'y écouler tant que l'invariant `x <= backoff_D` est vrai, `backoff_D` étant la durée de *backoff*. Depuis cet état, deux transitions sont possibles :

- une qui est prise au moment où `x == backoff_D` car l'invariant de l'état *backoff* oblige à prendre une transition (et la garde le permet). Cette transition correspond au fait que le nœud gagne la contention, il émet donc un signal de synchronisation pour le signifier ($c!$) ;

- l'autre transition correspond à la réception d'un signal de synchronisation avant la fin de la durée du *backoff*, la transition n'a pas de garde, elle est toujours active et est prise sur réception du signal. Ce cas correspond à la perte de la contention.

Cette modélisation est utilisée pour représenter les deux périodes de *backoff* de RTXP.

9.1.2 Modélisation des transmissions et collisions

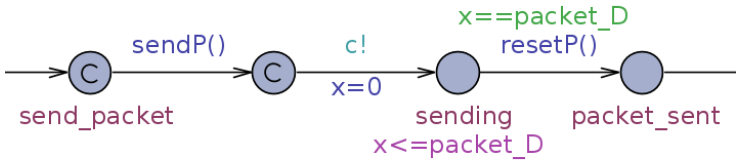
Pour la modélisation des transmissions, nous utilisons le modèle où la connectivité est vérifiée avant synchronisation comme préconisé par les résultats de l'étude de la section 8.2 du Chapitre 8. Nous adaptons cette modélisation à RTXP : nous utilisons un tableau de variables globales (au lieu d'une seule variable globale comme dans l'étude de la section 8.2) pour pouvoir représenter des communications simultanées dans le réseau, nous modélisons les transmissions de *jamming codes* qui sont détectables dans le 2-voisinage de l'émetteur et nous modélisons la file de paquets des nœuds.

Les collisions sont modélisées conjointement avec les transmissions puisqu'elles surviennent pendant les transmissions. Nous modélisons des interférences à deux sauts, c'est-à-dire que des nœuds éloignés de plus de deux sauts n'interfèrent pas entre eux et donc une collision se produit quand une transmission a lieu dans le 2-voisinage d'un nœud qui est déjà en train de recevoir un paquet d'un de ses voisins.

Transmission des paquets de données avec collisions

Les modèles d'émission et de réception pour RTXP sont donnés dans les Figures 9.2 et 9.3. Comme indiqué dans la section 8.2 du Chapitre 8, la topologie du réseau est représentée par une matrice d'adjacence. Pour RTXP, nous en définissons deux : une qui représente les connexions directes entres les nœuds (nommée `connect1`) et l'autre qui représente les connexions à deux sauts (nommée `connect2`). Le médium de communication est représenté par la variable canal *broadcast* c. Chaque nœud dispose d'une file d'attente `m` d'une capacité M paquets qui est représentée sous forme d'un tableau d'entiers : un paquet est identifié par un entier qui correspond à l'identifiant du nœud qui l'a émis pour la première fois. Les échanges de valeurs entre nœuds se font à l'aide d'un tableau d'entier `msgTab` (déclaré en variable globale) de taille égale au nombre de nœuds dans le réseau N. De cette manière plusieurs passages de valeurs peuvent avoir lieu simultanément dans le réseau : chaque nœud met à jour la case qui correspond à son identifiant (`msgTab[i]==1` signifie que le nœud i est en train de transmettre un paquet).

La Figure 9.2(a) représente la modélisation de l'émission d'un paquet dans le cas de RTXP. Le nœud, lorsqu'il a un paquet à émettre, passe dans l'état `send_packet`. Cet état est immédiatement suivi (car c'est un état *committed*) d'une transition qui met à jour des variables globales correspondantes au paquet. La mise à jour est réalisée par la fonction `sendP()`, décrite dans la Figure 9.2(b) : le message courant est pris au début de la file de messages (`m[0]`), ensuite la file est mise à jour, c'est-à-dire que les données sont toutes décalées d'une case vers la gauche et la dernière case est mise



(a) TA UPPAAL de l'émetteur

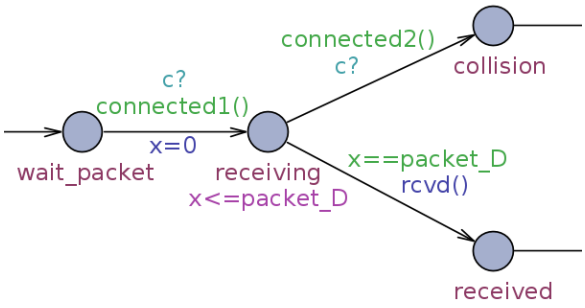
```
void sendP(){
    currentMsg=m[0];
    for(i=0;i<M-1;i++){
        m[i]=m[i+1];
    }
    m[M-1]=0
    msgTab[id]=currentMsg;
}

void resetP(){
    msgTab[id]=0;
}
```

(b) fonctions `sendP()` et `resetP()`

FIGURE 9.2 – Modélisation des transmissions de paquets : l'émetteur

à 0. Le nœud inscrit ensuite la valeur du message dans la case du tableau `msgTab` qui correspond à son identifiant. Une fois cette transition effectuée, une deuxième est immédiatement prise, elle permet d'émettre un signal de synchronisation sur c. Lors de la même transition, le nœud remet son horloge x à 0. Il se trouve ensuite dans l'état `sending` et y reste pour la durée d'émission du paquet `packet_D`, il passe ensuite la dernière transition vers l'état `packet_sent` qui remet à 0 la case du tableau `msgTab`. On note que l'émetteur ne s'occupe pas des collisions, car elles surviennent au niveau du récepteur.

La figure 9.3(a) représente le modèle de réception de paquets pour le cas de RTXP. Lorsqu'un nœud est dans l'état `wait_packet`, il est près à recevoir un paquet de données. Il attend un signal sur la variable canal *broadcast* c. Lors de la réception d'un signal, le nœud peut se synchroniser seulement si la fonction `connected1()` retourne la valeur VRAI. Cette fonction permet de vérifier que le nœud est voisin de l'émetteur, elle est représentée sur la figure 9.3(b) : une boucle parcourt le tableau `msgTab`, si une case dont la valeur est différente de zéro est trouvée (un nœud émet une donnée) et que le nœud correspondant est un voisin (si la valeur de la matrice d'adjacence `connect1[id][i]`, est supérieure à zéro, cela signifie que les nœuds sont connectés) alors la fonction retourne la valeur VRAI et la transition vers l'état `receiving` peut être prise. Cet état correspond à une réception en cours. Deux cas sont alors possibles : soit la réception finie normalement, soit un autre paquet arrive ce qui provoque une collision. Dans le premier cas, il n'y a pas de synchronisation durant toute la durée du paquet `packet_D` : la réception est donc un succès et la fonction `rcvd()` décrite dans la Figure 9.3(c) permet d'ajouter le paquet reçu dans le premier emplacement libre de



(a) TA UPPAAL du récepteur

```
bool connected1(){
    int i;
    for(i=0;i<N;i++){
        if(connect1[id][i] && msgTab[i]!=0){
            return true;
        }
    }
    return false;
}

bool connected2(){
    int i;
    int nb=0;
    for(i=0;i<N;i++){
        if(connect2[id][i] && msgTab[i]!=0){
            nb=nb+1;
        }
    }
    if(nb<=1){
        return false;
    }else{
        return true;
    }
}
```

(b) fonctions `connected1()` et `connected2()`

```
void rcvd(){
    int i;
    for(i=0;i<N;i++){
        if(connect1[id][i] && msgTab[i]!=0){
            currentMsg=msgTab[i];

        }
    }

    int j;
    bool found=0;
    for(j=0;j<M-1;j++){
        if(!found && m[j]==0){
            m[j]=currentMsg;
            found=1;
        }
    }
}
```

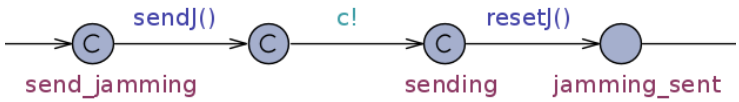
(c) fonction `rcvd()`

FIGURE 9.3 – Modélisation des transmissions de paquets : le récepteur

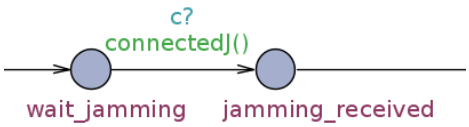
la file d'attente du nœud, lors du passage de la transition vers l'état `received`. Dans le second cas, qui correspond à une collision, une synchronisation a lieu depuis l'état `receiveing`, la synchronisation est possible si la fonction `connected2()`, représentée dans la Figure 9.3(b), retourne la valeur VRAI. Cette fonction vérifie si un nœud dans le 2-voisinage émet un paquet : son fonctionnement est similaire à celui de la fonction `connected1()` mais c'est le tableau `connect2` qui est vérifié et donc les connexions à deux sauts, car l'émission d'un nœud du 2-voisinage provoque une collision (modèle d'interférences à deux sauts) ; de plus, on vérifie dans la fonction `connected2()` qu'il y a deux émetteurs dans le 2-voisinage grâce à la variable `nb` et non un comme dans `connected1()`, car il faut au moins deux paquets pour avoir une collision (celui que l'on est en train de recevoir et un autre).

Transmission des *jamming codes*

La transmission des *jamming codes* est plus simple à modéliser que celle des paquets de données : il n'y a pas besoin de modéliser les collisions car les messages ne transportent pas d'information explicite et il n'y a pas besoin de modéliser de file de message. Comme indiqué dans la section 5.2 du chapitre 5, les *jamming codes* sont détectés par les nœuds du 2-voisinage de l'émetteur. Le tableau de booléens déclaré en variable globale `jammingTab` est utilisé pour indiquer quels nœuds émettent un *jamming code*.



(a) TA UPPAAL de l'émetteur



(b) TA UPPAAL du récepteur

```
void sendJ(){
    jammingTab[id]=0;
}

void resetJ(){
    jammingTab[id]=0;
}
```

(c) fonctions `sendJ()` et `resetJ()`

```
bool connectedJ(){
    int i;
    for(i=0;i<N;i++){
        if(connect2[id][i] && jammingTab[i]){
            return true;
        }
    }
    return false;
}
```

(d) fonction `connectedJ()`

FIGURE 9.4 – Modélisation des transmissions de *jamming codes*

La Figure 9.4 illustre la modélisation UPPAAL des transmissions de *jamming codes*. La Figure 9.4(a) représente le TA de l'émetteur, il est très similaire à celui du cas des paquets de données de la Figure 9.2 : le tableau `jammingTab` est d'abord mis à jour à l'aide de la fonction `sendJ()` représenté sur la Figure 9.4(c), ensuite un signal est émis sur la variable canal *broadcast* c. Contrairement au cas de l'émission de paquets, le nœud n'attend pas dans l'état `sending` : on modélise l'émission d'un *jamming code* comme étant instantanée, car les collisions ne sont pas représentées.

L'émetteur, dont le TA est illustré par la figure 9.4(b), est plus simple que pour le cas des transmissions de paquets car les collisions ne sont pas gérées. Le nœud attend une synchronisation dans l'état `wait_jamming`. Si un signal est émis sur la variable canal *broadcast* c et que la fonction `connectedJ()`, représentée dans la Figure 9.4(d),

retourne VRAI, alors la transtion est prise vers l'état `jamming_received`. La fonction `connectedJ()` consiste à vérifier qu'il y a un nœud qui émet un *jamming code*, c'est-à-dire qu'une des cases du tableau `jammingTab` est VRAI et que le nœud correspondant est dans le 2-voisinage. Dans la section suivante, nous présentons le modèle complet de RTXP, qui est vérifié dans les sections suivantes.

9.1.3 Modèle global de RTXP

Le modèle TA UPPAAL de RTXP est illustré par la Figure 9.5 ; les déclarations de variables et fonctions, globales et locales sont détaillées dans l'Annexe 10.2.2. Les phases de RTXP correspondent aux zones colorées de la Figure 9.5, on différencie le cas où le nœud émet un paquet du cas où il en reçoit un. Les phases de RTXP sont détaillées dans la section 5.5 du Chapitre 5, elles sont modélisées de la manière suivante :

- La phase B correspond au *backoff* pour gagner l'accès au canal, seuls les nœuds qui ont un paquet à émettre l'exécutent. Le modèle de la phase B correspond à un *backoff* et à la transmission d'un *jamming code* si le nœud gagne la contention, comme représentés sur les figures 9.1 et 9.4.
- La phase R correspond à la transmission d'un paquet. Le modèle pour un émetteur (noté `R-émetteur` sur la Figure 9.5) correspond à la figure 9.2(a), pour un récepteur (noté `R-récepteur`) c'est la Figure 9.3.
- La phase BF correspond au *backoff* pour relayer un paquet, l'émetteur du paquet attend un *jamming code* les nœuds récepteurs entrent en contention. Pour l'émetteur du paquet de données, cette phase correspond donc à la réception d'un *jamming code* comme modélisé dans la Figure 9.4(b). Pour le récepteur (du paquet de données) cela correspond à un *backoff* et à un *jamming code* si le nœud gagne la contention, comme représentés sur les figures 9.1 et 9.4.
- Le *slot* L correspond au *slot* qui permet aux nœuds qui ont perdu la phase B de déclencher une période d'activité secondaire. Cette phase correspond à la transmission d'un *jamming code* (Figure 9.4).

Chaque nœud est initialisé avec les paramètres suivants : son identifiant ID, une variable qui définit s'il a un paquet à envoyer au début de la simulation `mts` et sa coordonnée, présenté dans le Chapitre 4 (décomposée en gradient `n` et `offset`). Les variables `backoffD`, `rcvSlot` et `jamming` correspondent respectivement aux durées des phases B (et BF), R et L. La variable `Lheard` indique au nœud s'il exécute une période d'activité principale ou secondaire, la variable DC correspond au rapport de cycle d'endormissement. L'ordre d'exécution des phases dépend du paramètre n mod 3 comme détaillé dans la section 5.5 du Chapitre 5. Ce paramètre est vérifié grâces à des conditions dans les gardes des transitions permettant de passer d'une phase à l'autre. Les états et transitions qui ne sont pas dans les zones colorées (de la Figure 9.5) permettent d'ordonnancer l'enchaînement des phases pour les différents nœuds du réseau. Par exemple, l'état `waitL` permet aux nœuds qui ont fini d'exécuter leurs phases B, R et BF d'attendre le *slot* L.

Le modèle de la Figure 9.5 est une base qui est utilisé pour vérifier RTXP, d'abord en ne prenant pas en compte les liens non-fiables, puis en utilisant la technique de

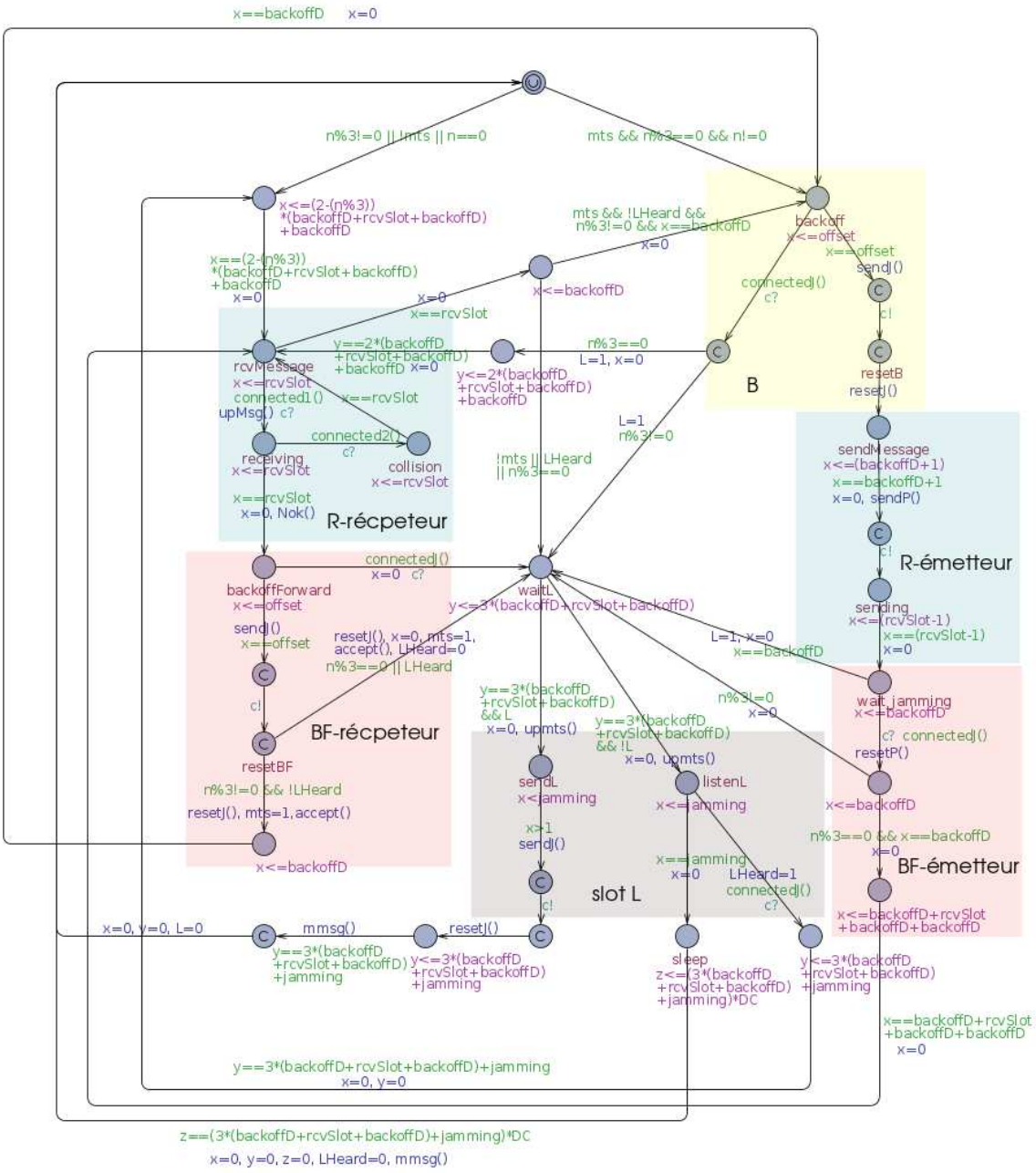


FIGURE 9.5 – Modélisation de RTXP en TA UPPAAL

vérification incluant le lien probabiliste développée dans la section 8.3 du Chapitre 8, et enfin pour vérifier RTXP sur des réseaux de tailles plus importantes avec la méthode hybride proposée dans la section 8.4 du Chapitre 8. Pour chacun de ces cas, le modèle est adapté (ces adaptations sont détaillées dans les sections suivantes), mais la structure générale reste la même. Ces études nous permettent, d'une part de vérifier formellement le protocole RTXP et d'autre part de montrer l'applicabilité des techniques développées dans le Chapitre 8 à l'étude d'un protocole temps-réel pour RCsF.

9.2 Vérification temporelle

Dans cette section, nous vérifions le protocole RTXP en prenant l'hypothèse que la seule source de perte de paquets sont les collisions. Cela nous permet de nous assurer du bon fonctionnement du protocole avec une propagation idéale (pas de *fading*, seulement des pertes en espace libre). Trois types de topologies sont étudiés :

- une clique : tous les nœuds sont dans le même voisinage, cela permet de faire varier le degré d'un voisinage ;
- une ligne dont une des deux extrémités est le puits, cela permet de faire varier le diamètre du réseau ;
- une topologie aléatoire uniforme, cela permet de faire varier à la fois le degré et le diamètre du réseau ;

Chaque nœud à un paquet à envoyer au puits, les paquets doivent arriver avant l'échéance définie par l'Equation 5.6 de la section 5.6 du chapitre 5 (p.93). Cette propriété est exprimée en langage UPPAAL de la manière suivante :

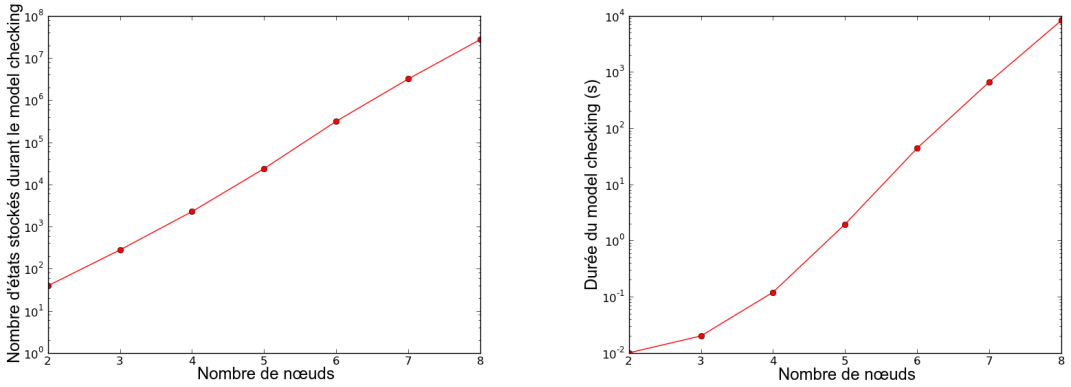
$$\texttt{A[]}\ \texttt{sr} == \texttt{M}\ \texttt{imply}\ \texttt{a} \leq \texttt{N} * (3 * (\texttt{backoffD} + \texttt{rcvSlot} + \texttt{backoffD}) + \texttt{jamming}) * \texttt{DC} \tag{9.1}$$

avec `a` une horloge qui n'est jamais remise à zéro, elle permet de mesurer le temps écoulé depuis la configuration initiale ; `sr` le nombre de paquets reçus par le puits ; `M` le nombre de nœuds sources dans le réseau (tous les nœuds sauf le puits sont des sources) ; `N` le nombre maximum de sauts pour atteindre le puits et $(3 * (\texttt{contentionD} + \texttt{rcvSlotD} + \texttt{contentionD}) + \texttt{jammingD}) * \texttt{DC}$ la durée d'un cycle d'endormissement. Cette propriété signifie donc que dans toutes les configurations accessibles depuis la configuration initiale, le fait que le puits ait un nombre de paquets reçus égal au nombre de sources implique qu'il s'est écoulé moins de `N` cycles d'endormissement (dans le modèle, `sr` est remis à zéro dès qu'il atteint `M` sinon la propriété est toujours violée à partir du moment où le puits a reçu tous les paquets et que le temps continue à s'écouler).

Pour chaque type de topologie, nous faisons varier le nombre de nœuds à partir de 2 et avec un pas de 1 jusqu'à ce que le *Model Checking* ne soit plus réalisable (espace de configuration de taille plus importante que les 4Go de la machine utilisée pour réaliser la vérification).

9.2.1 Topologies cliques

Nous vérifions d'abord RTXP sur des topologies où tous les nœuds sont voisins du puits et voisins entres eux. Tous les nœuds sources sont donc à un saut du puits, leurs paquets doivent par conséquent atteindre le puits en un cycle d'endormissement au maximum. Ce type de topologie nous permet de détecter les erreurs de modélisation dans les transmissions des nœuds (erreur sur les conditions de réceptions de paquet ou de collisions). La taille des topologies vérifiées varie de 2 à 8 nœuds, au delà l'espace d'états généré ne tient plus dans la mémoire de la machine utilisée pour la vérification.



(a) Nombre de configurations stockée en fonction du nombre de nœuds

(b) Durée du *Model Checking* en fonction du nombre de nœuds

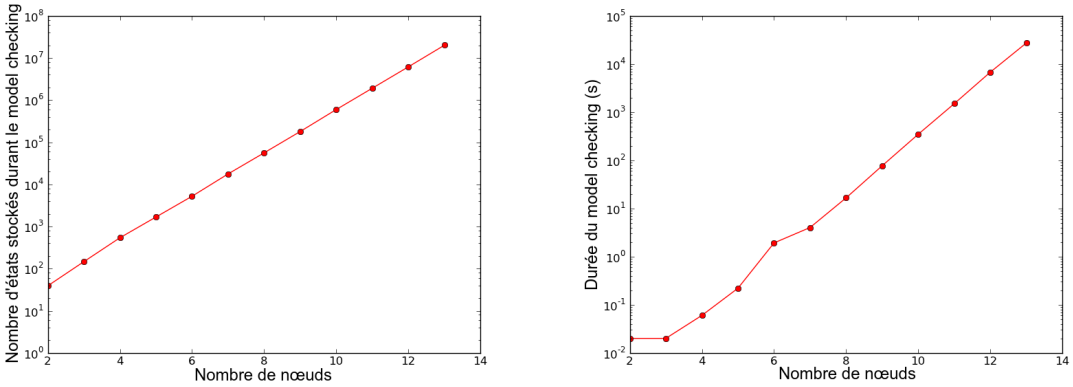
FIGURE 9.6 – Performances du *Model Checking* dans le cas des topologies cliques

Les Figures 9.6(a) et 9.6(b) représentent respectivement le nombre de configurations stockées et la durée du *Model Checking* en fonction du nombre de nœuds dans le réseau pour le modèle de la Figure 9.5. Pour toutes les vérifications effectuées ici la propriété est vraie (tous les paquets atteignent le puits en un cycle d'endormissement). Pour les deux graphes, l'échelle des ordonnées est logarithmique, on constate (comme dans la section 8.2.3 du Chapitre 8 pour GRAB) que l'espace de configurations et la durée de *Model Checking* augmentent exponentiellement avec le nombre de nœuds dans le réseau. On note que même pour des espaces de configuration qui remplissent quasiment totalement la mémoire disponible (cas des réseaux de 8 nœuds), la durée de vérification reste raisonnable : quelques heures seulement.

9.2.2 Topologies linéaires

Pour les topologies linéaires, chaque nœud (sauf les extrémités de la ligne) a seulement deux voisins : un en direction du puits et un dans la direction opposée. Cette topologie nous permet de déboguer la modélisation des transmissions multi-sauts sur le cas le plus simple possible.

Les Figures 9.7(a) et 9.7(b) représentent respectivement le nombre de configurations stockées et la durée du *Model Checking* en fonction du nombre de nœuds dans le réseau pour le modèle de la Figure 9.5, avec une échelle des ordonnées logarithmique. On constate que cette fois il est possible de vérifier un réseau de 13 nœuds au maximum, cette augmentation de la taille des réseaux vérifiables est dû au fait qu'avec la topologie ligne un nœud a moins de voisins que dans le cas précédent. Il y a donc moins d'interactions entre les nœuds et donc moins d'entrelacements lors de la transmission des paquets et *jamming codes*.



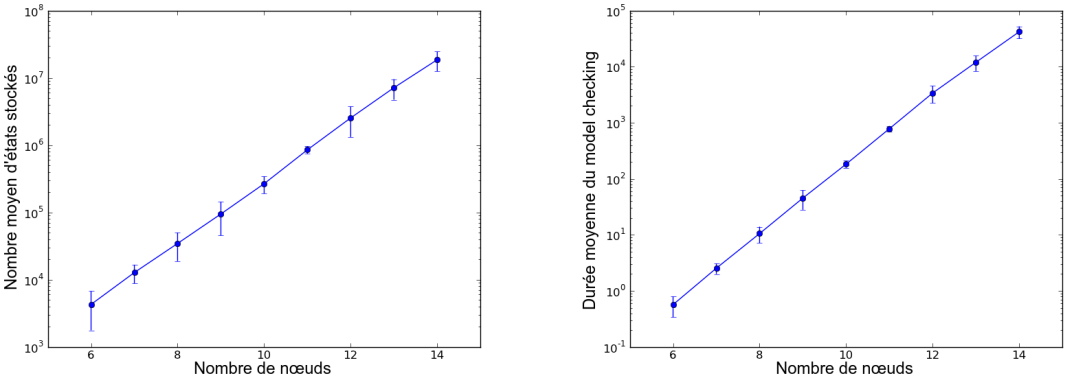
(a) Nombre de configurations stockée en fonction du nombre de nœuds

(b) Durée du *Model Checking* en fonction du nombre de nœuds

FIGURE 9.7 – Performances du *Model Checking* dans le cas des topologies lignes

9.2.3 Topologies aléatoires uniformes

Nous nous intéressons maintenant à des topologies aléatoires. Comme dans le cas de la section 8.2.3 du Chapitre 8, nous considérons des topologies générées aléatoirement de manière uniforme de 5 à 20 nœuds sources (plus un puits). Pour chaque nombre de nœuds, 10 topologies sont générées. Ce type de topologie permet de déboguer les cas de routage multi-sauts, mais avec des voisinages de plus de deux nœuds contrairement au cas précédent.



(a) Nombre de configurations stockée en fonction du nombre de nœuds

(b) Durée du *Model Checking* en fonction du nombre de nœuds

FIGURE 9.8 – Performances du *Model Checking* dans le cas des topologies aléatoires

Les Figures 9.8(a) et 9.8(b) représentent respectivement le nombre moyen de configurations stockées et la durée du *Model Checking* en fonction du nombre de nœuds dans le réseau pour le modèle de la Figure 9.5, avec une échelle des ordonnées logarithmique. Les barres d'erreurs représentent deux fois la valeur de l'écart type. Dans

ce cas nous sommes capables de vérifier des réseaux de 14 nœuds au maximum (au delà la mémoire disponible n'est pas suffisante), la durée de vérification pour les topologies de 14 nœuds varie entre 10 et 15 heures. Dans ce cas également, on ne détecte pas d'erreur dans RTXP.

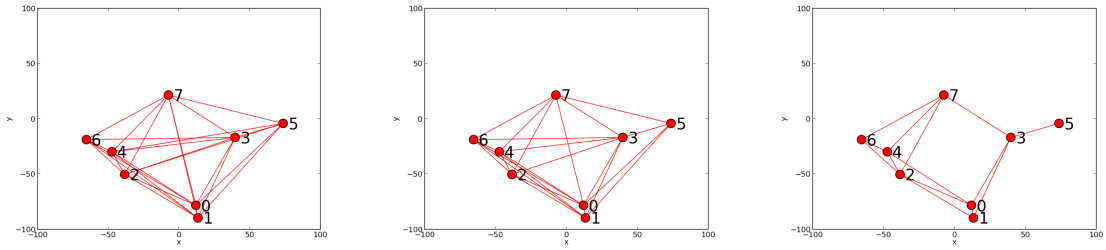
Les cas présentés dans cette section permettent de corriger le modèle du protocole étudié. Cela ne permet cependant pas de vérifier des RCsF réalistes pour lesquels les pertes de paquets ne sont pas seulement dues à des collisions et qui peuvent être composés de plusieurs centaines de nœuds. Dans les sections suivantes nous appliquons les techniques des sections 8.3 et 8.4, pour prendre en compte la non-fiabilité du lien radio et vérifier des réseaux plus grands.

9.3 Prise en compte des liens non-fiables

Dans cette section nous appliquons la méthode de vérification avec prise en compte des liens non-fiables présentée dans la section 8.3 du Chapitre 8 à RTXP. Pour automatiser le processus de vérification, nous avons implémenté en Python les algorithmes 8.3.1 et 8.3.2 (respectivement p.168 et p.170). Nous proposons de vérifier RTXP sur une topologie physique arbitraire (les positions des nœuds on été tirées aléatoirement) à partir de laquelle nous générons plusieurs topologies de base comme détaillé dans la section suivante.

9.3.1 Cas d'étude

La Figure 9.9 représente les 3 topologies de base que nous générons à partir de la même topologie physique. Ces différentes topologies sont obtenues en faisant varier le seuil P_{th} au-delà duquel le lien est considéré comme existant. Nous choisissons les seuil 0.7, 0.8 et 0.9, les topologies de base correspondantes sont représentées respectivement par les Figures 9.9(a), 9.9(b) et 9.9(c).



(a) $P_{th} = 0.7$ (b) $P_{th} = 0.8$ (c) $P_{th} = 0.9$

FIGURE 9.9 – Topologies de base pour la vérification de RTXP

Plus le seuil est petit, plus la topologie comporte de liens : pour un seuil de 0.7 tous les liens sur lesquels la probabilité de réception d'un paquet est supérieure à 0.7 sont considérés alors que pour un seuil de 0.9 on considère seulement ceux où elle est supérieure à 0.9. L'Equation 3.8 (p.45) est utilisée pour calculer la probabilité de

réception d'un paquet, les paramètres utilisés sont ceux du tableau 3.1 (respectivement p.45 et 44). Lors de la vérification, on fait varier le nombre de nœuds de 3 à 8 par pas de 1 (par exemple, pour la topologie de trois nœuds, on ne considère que les nœuds 0, 1 et 2).

Le modèle de RTXP de la Figure 9.5 est donné en entrée de l'algorithme de vérification. Dans les scénarios étudiés, chaque nœud possède un paquet à envoyer. Comme indiqué dans la section 8.3 du Chapitre 8, tous les paquets doivent effectuer un saut en direction du puits dans un cycle d'endormissement, nous vérifions donc les propriétés suivantes (données ici pour un réseau de 8 nœuds) :

$$\begin{aligned} &\texttt{A} <> (\texttt{msgR}[0] == 1 \texttt{ and } \texttt{msgR}[1] == 1 \texttt{ and } \texttt{msgR}[2] == 1 \texttt{ and } \texttt{msgR}[3] == 1 \\ &\texttt{and } \texttt{msgR}[4] == 1 \texttt{ and } \texttt{msgR}[5] == 1 \texttt{ and } \texttt{msgR}[6] == 1 \texttt{ and } \texttt{msgR}[7] == 1) \end{aligned} \quad (9.2)$$

$$\begin{aligned} &\texttt{A } [] (\texttt{msgR}[0] == 1 \texttt{ and } \texttt{msgR}[1] == 1 \texttt{ and } \texttt{msgR}[2] == 1 \texttt{ and } \texttt{msgR}[3] == 1 \\ &\texttt{and } \texttt{msgR}[4] == 1 \texttt{ and } \texttt{msgR}[5] == 1 \texttt{ and } \texttt{msgR}[6] == 1 \texttt{ and } \texttt{msgR}[7] == 1) \\ &\texttt{imply a} \leq \texttt{N} * (3 * (\texttt{backoffD} + \texttt{rcvSlot} + \texttt{backoffD}) + \texttt{jamming}) * \texttt{DC} \end{aligned} \quad (9.3)$$

La valeur $\texttt{msgR}[\texttt{i}] == 1$ indique que le message du nœud i a été reçu par un voisin plus proche du puits. La propriété 9.2 permet de vérifier que "quel que soit le chemin d'exécution pris, tous les paquets font un saut en direction du puits". La propriété 9.3 permet de vérifier que "tous les paquets font ce saut en moins d'un cycle d'endormissement".

9.3.2 Résultats et performances de vérification

Dans cette section, nous présentons et commentons les résultats et les performances de vérification de RTXP avec des liens non-fiables.

Les résultats sont résumés dans le Tableau 9.1, la première colonne indique la taille de la topologie vérifiée. Les paramètres évalués sont : le nombre de topologies vérifiées, la durée de vérification et la probabilité que les propriétés 9.2 et 9.3 soient vraies. Ces paramètres sont évalués sur les trois topologies de base correspondant aux seuils P_{th} 0.7, 0.8 et 0.9.

On constate que pour la topologie de 3 nœuds, la valeur du seuil P_{th} n'a pas d'influence sur les performances et les résultats de vérification. En effet, les topologies vérifiées sont les mêmes dans les trois cas : ce sont les topologies pour lesquelles les nœuds sont directement connectés au puits avec le lien entre les nœuds 1 et 2 (on constate que sur la Figure 9.9 que le seuil n'altère pas la topologie formée par les nœuds 0, 1 et 2).

Pour les topologies de 4 et 5 nœuds, les résultats de nombre de topologies vérifiées et durées pour $P_{th} = 0.7$ et $P_{th} = 0.8$ sont les mêmes car les topologies de base sont les mêmes avec ces seuils. En revanche pour $P_{th} = 0.9$ des liens sont éliminés de la topologie de base, cela implique de vérifier moins de topologie et réduit donc la durée

Nombre de nœuds	Nombre de topologies vérifiées			Durée (s)			P_{pv}		
	0.7	**0.8**	**0.9**	**0.7**	**0.8**	**0.9**	**0.7**	**0.8**	**0.9**
3	2	2	2	1.380	0.465	0.433	0.98	0.98	0.98
4	8	8	4	1.430	1.432	0.779	0.93	0.93	0.93
5	64	64	8	14.827	14.962	1.952	0.85	0.85	0.85
6	1024	256	8	1370.189	165.596	3.366	0.69	0.69	0.85
7	-	-	24	-	-	27.846	-	-	0.85
8	-	-	336	-	-	1729.765	-	-	0.85

TABLE 9.1 – Résultats de vérification de RTXP avec des liens non-fiables

de vérification. Au niveau des résultats de vérification, on observe que les probabilités P_{pv} sont les mêmes avec les trois valeurs de P_{th}, c'est dû au fait que dans les trois cas tous les nœuds sont directement connectés au puits et que donc P_{pv} correspond à la probabilité que tous les paquets soient reçus par le puits. En revanche, ce n'est plus le cas pour la topologie de 6 nœuds, en effet, on constate sur la Figure 9.9 que dans les cas $P_{th} = 0.7$ et $P_{th} = 0.8$ le nœud 5 est directement relié au puits par un long lien alors que dans le cas $P_{th} = 0.9$ il est seulement relié au nœud 3 qui lui-même est relié au puits. Le paquet du nœud 5 doit alors effectuer un saut sur un lien plus court donc plus fiable, cela explique l'augmentation de P_{pv} dans le cas $P_{th} = 0.9$. Pour la topologie de 6 nœuds, on constate aussi que plus P_{th} est petit, plus il y a d'arêtes dans la topologie de base donc plus le nombre de cas à vérifier est grand (donc la durée de vérification augmente). Cette augmentation du nombre d'arêtes et donc de cas à vérifier rend impossible la vérification des topologies de 7 et 8 nœuds pour les seuils $P_{th} = 0.7$ et $P_{th} = 0.8$.

On observe, avec ce cas d'étude, qu'il est possible de vérifier des réseaux de huit nœuds au maximum. L'augmentation de la durée de vérification est, avec cette méthode de vérification, exponentielle en fonction du nombre de d'arêtes. On peut donc vérifier des réseaux plus grands s'ils comportent un nombre limité d'arêtes (on note tout de même que pour que le réseau reste connecté, il faut au minimum que le nombre d'arêtes augmente linéairement avec le nombre de nœuds). Par exemple, la Figure 9.10 montre un réseau de 10 nœuds (et 13 arêtes) pour lequel 64 topologies ont été vérifiées en 2h52m8s.

Bien que le passage à l'échelle de cette méthode reste limité, elle est néanmoins applicable aux réseaux de capteurs biomédicaux qui comportent peu de nœuds (une dizaine au maximum [IEEE, 2012]). Dans la section suivante, nous appliquons la méthode de passage à l'échelle de la vérification formelle présentée dans le chapitre précédent (section 8.4) au cas de RTXP.

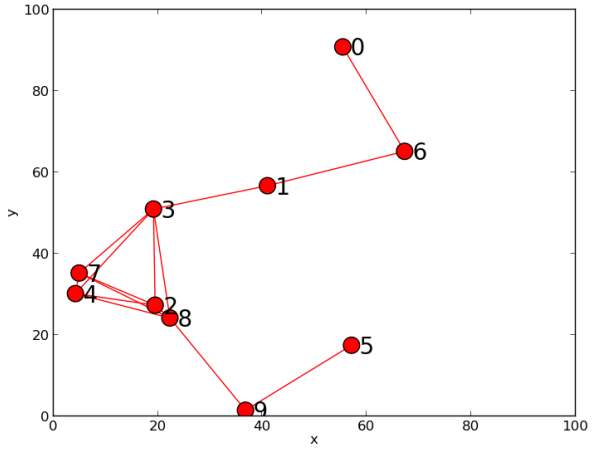


FIGURE 9.10 – Réseau de 10 nœuds vérifié en tenant compte des liens non-fiables

9.4 Passage à l'échelle

Dans cette section, nous appliquons la méthode hybride *Network Calculus* et *Model Checking* (développée dans la section 8.4 du Chapitre 8) à RTXP. Cela nous permet de vérifier le protocole sur de plus grands réseaux qu'avec les méthodes précédentes.

Pour réaliser les vérifications, l'Algorithme 8.4.1 (p.193) a été implémenté* en Python. Nous rappelons que cet algorithme permet de vérifier que chaque nœud du réseau est bien capable de servir son trafic (représenté sous formes de courbes d'arrivées). L'implémentation des opérations de *Network Calculus* et la représentation des courbes sous forme de structures de données sont issues de [Wandeler, 2006]. L'Algorithme 8.4.1 prend en entrée la topologie du réseau sous forme d'arbre enraciné au puits, l'arbre représente les chemins de routage produits par le protocole étudié. Dans le cas de RTXP, pour produire cet arbre, chaque nœud prend comme parent son voisin le plus proche du puits selon la coordonnée virtuelle, car c'est celui qui relaye ses paquets (on rappelle qu'il est difficile de représenter un routage dynamique en *Network Calculus* [Jouhet, 2012] et que notre méthode hybride ne le permet pas). Le modèle de RTXP de la Figure 9.5 est également fourni en entrée.

L'algorithme prend aussi en entrée une hypothèse sur le service que sont capables de fournir les nœuds et sur le trafic généré par chaque nœud : pour le service, nous supposons qu'un nœud est capable de servir tous les paquets présents dans sa file de messages au début d'un cycle d'endormissement avant la fin de ce cycle (le nombre de paquets dépend des hypothèses sur les arrivées). Cela correspond au délai maximum défini pour réaliser un saut défini dans la section 5.6 du Chapitre 5 (la courbe de service correspondante a la forme illustrée par la figure 8.19 dans le chapitre précédent p.183) ; pour le trafic, on propose d'associer à chaque nœud une courbe d'arrivées sur les alarmes détectées comme expliqué dans la section 8.4.3 du chapitre précédent.

∗. Le code de l'outil est visible aux adresses suivantes : `http://perso.citi.insa-lyon.fr/amouradia/code/verification/scalableChecker.py`

Comme indiqué dans la section 8.4, pour chaque nœud du réseau deux TA UPPAAL sont produits : un qui représente le trafic des fils du nœud et un qui représente les compétiteurs prioritaires du nœud, ces TA représentent les interactions du nœud avec le réseau. Le détail de leur construction est donné dans la section suivante.

On note par ailleurs que l'on utilise l'amélioration présentée dans la section 8.4.4 du Chapitre 8 qui consiste à vérifier uniquement les nœuds dont les courbes d'arrivées ne sont pas strictement inférieures aux autres nœuds.

9.4.1 Implémentation des TA représentant les interactions du réseau avec le nœud vérifié

Dans cette section, nous décrivons comment sont construits les TA qui représentent les interactions d'un nœud avec le reste du réseau. C'est un point clef de notre technique de vérification hybride, car c'est cela qui permet de réduire l'explosion de l'espace d'états en abstrayant la complexité du réseau.

Représentation des interactions avec les nœuds fils

Lors de l'implémentation de l'algorithme de vérification, nous avons exploré deux manières différentes de représenter les interactions d'un nœud avec ses fils : soit par mise à jour direct de la file de messages du nœud, soit en représentant toutes les réceptions de paquets explicitement.

- Avec la première solution, les fils sont représentés par un TA qui correspond à la Figure 8.23 (p.189) : sur prise de la transition qui correspond à un événement, il met à jour une variable globale qui correspond au nombre de paquets que le nœud doit relayer. Cette solution masque certains comportements (réceptions de paquets et émissions de *jamming codes* qui correspondent aux parties R-récepteur et BF-récepteur sur la Figure 9.5) où des erreurs pourraient se produire, mais cela permet de réduire le temps de vérification d'un nœud.
- Avec la seconde solution, toutes les interactions du nœud avec ses fils dans l'arbre de routage sont fidèlement représentées. On utilise alors deux TA : un qui correspond à la Figure 8.23 (p.189). Il a le même comportement que dans le cas précédent mais au lieu d'incrémenter une variable globale correspondant à la file d'attente du nœud, il incrémente une variable d'un deuxième automate qui émet les paquets. Ce deuxième automate correspond aux parties B, R-émetteur, BF-émetteur et *slot* L de la Figure 9.5. La durée de vérification d'un nœud, dans ce cas, augmente fortement car le nombre d'horloges utilisées est plus grand que dans le premier cas.

Nous adoptons donc la première solution pour garder des temps de vérification raisonnables (de l'ordre de quelques dizaines d'heures au maximum). On note que cette représentation est acceptable, dans la mesure où le trafic des fils dans le cas de RTXP n'influence pas la capacité du nœud à écouler son trafic (un nœud n'est pas en compétition pour le canal avec ses fils).

Représentation des interactions avec les compétiteurs prioritaires

Dans le cas de RTXP, les compétiteurs prioritaires d'un nœud sont ceux qui entrent en contention avec ce nœud dans la phase B et qui ont une coordonnée virtuelle inférieure à la sienne. Les interactions avec les compétiteurs prioritaires empêchent le nœud d'écouler son propre trafic et influent donc sur sa capacité de service. Dans ce cas, il faut donc représenter explicitement le trafic des compétiteurs pour être sûr que le nœud vérifié est capable de servir le trafic qu'il produit et qu'il reçoit de ses fils. On utilise donc la méthode de représentation avec deux TA : un qui émet des arrivées (Figure 8.23 p.189) et met à jour une variable globale qui correspond à la file d'attente d'un deuxième TA qui se comporte comme les compétiteurs prioritaires (cela correspond aux parties B -avec une coordonnée plus petite que le nœud vérifié- et *slot* L du modèle de la Figure 9.5).

Ces automates sont donnés en entrée de l'algorithme de vérification et sont paramétrés pour chaque nœud en fonction des courbes d'arrivées associées au nœud (les courbes d'arrivées sont calculées grâces aux lignes 3 à 10 de l'Algorithme 8.4.1, le paramétrage de l'automate de la Figure 8.23 consiste à renseigner les valeurs des constantes `BMAX` et `delta`).

Dans la section suivante, nous présentons les scénarios sur lesquels nous appliquons la méthode de vérification ainsi que les performances et résultats de vérification.

9.4.2 Vérification

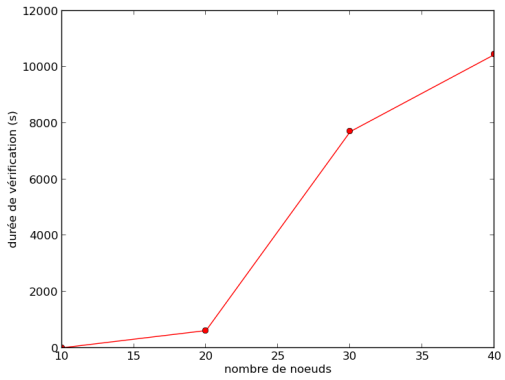
D'abord, nous générons des topologies aléatoires de 10, 20, 30 et 40 nœuds qui sont tous des sources d'alarmes pour observer les performances de la méthode de vérification lorsque la taille du réseau augmente. Nous vérifions ensuite un réseau de 500 nœuds avec 10, 20, 30 et 40 nœuds sources pour montrer que la durée de vérification dépend du trafic et non de la taille du réseau.

La propriété vérifiée est :

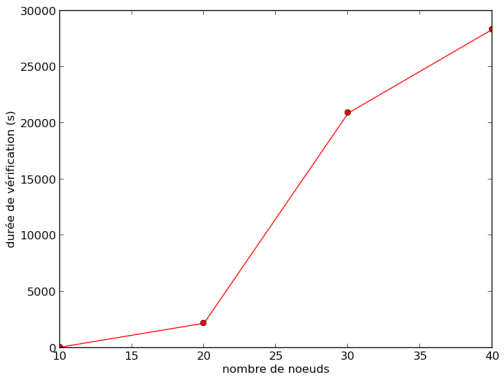
$$\texttt{A[] not N0.violation} \tag{9.4}$$

qui stipule que dans aucune configuration, le nœud vérifié (nommé `N0`) ne se trouve dans l'état `violation`. Cet état correspond à un dépassement de la capacité de RTXP exprimée par l'Equation 5.7 de la section 5.6 dans le Chapitre 5 (p.94).

La Figure 9.11(a) représente la durée de vérification des topologies où tous les nœuds sont des sources. Pour les topologies de 10, 20 et 30 nœuds, on vérifie bien que les nœuds sont capables de servir leur trafic. En revanche, pour la topologie de 40 nœuds, une trace produite par UPPAAL montre qu'un des nœuds dépasse la capacité de RTXP. C'est dû au fait que le nombre de paquet qu'il doit relayer durant un cycle d'endormissement est trop important. On peut régler ce problème soit en augmentant la durée du cycle d'endormissement (mais cela augmente le délai de bout en bout, comme mis en évidence par la Figure 5.4 p.95 du Chapitre 5) soit en modifiant la topologie pour que le trafic se répartissent entre plus de nœuds. On observe sur la Figure 9.11(a) que la durée de vérification augmente en fonction du



(a) Cas où tous les nœuds sont sources



(b) Réseau de 500 nœuds où le nombre de sources varie

FIGURE 9.11 – Durée de vérification en fonction du nombre de nœuds sources

nombre de nœuds dans le réseau. Cette augmentation est en fait due à l'augmentation du trafic induite par l'ajout de nouvelles sources : la durée de vérification augmente exponentiellement avec le nombre de sources. En effet, l'augmentation du trafic induit des courbes d'arrivée plus hautes (on rappelle que les courbes d'arrivées ont la forme de l'Equation 8.20 p.188, une courbe plus haute correspond à N plus grand et δ plus petit), l'augmentation de la courbe d'arrivée relâche les contraintes sur les trafics admissibles dans le nœud, la vérification de ces trafics possibles est donc plus longue (car il y a plus de cas à vérifier).

Nous vérifions que l'augmentation de la durée de vérification est bien due à l'augmentation du trafic et non à celle du nombre de nœuds dans le réseau en étudiant un réseau de 500 nœuds généré aléatoirement dans lequel nous faisons varier le nombre de sources de 10 à 40 avec un pas de 10 (les nœuds sources sont sélectionnés arbitrairement ; par exemple, pour 10 sources, les nœuds dont l'identifiant est un multiple de 50 sont sélectionnés). On observe sur la Figure 9.11(b) que malgré le grand nombre de nœuds du réseau, les durées de vérification restent comparables à celles de la Figure 9.11(a) : même si les valeurs doublent, on ne constate pas d'explosion de la durée (alors qu'on multiplie par plus de 10 la taille du réseau). Dans ce cas également pour les topologies avec 10, 20 et 30 sources, les nœuds sont capables de servir leur trafic. En revanche, comme dans le cas précédent, avec 40 sources, la capacité de RTXP est dépassée pour un des nœuds (cela explique l'inflexion de la fin de la courbe : UPPAAL termine avant d'avoir exploré tous les cas, car il trouve un chemin qui mène dans un "mauvais" état).

Ces résultats montrent l'applicabilité de la méthode de vérification hybride à des réseaux de grandes tailles, la limite vient du trafic. On constate tout de même que la vérification de nœuds dont le trafic est trop important pour RTXP reste possible.

9.5 Conclusion

Dans ce chapitre, nous présentons une application de la méthodologie de vérification des RCsF proposée dans le chapitre précédent. Nous commençons par décrire la modélisation des différents aspects de RTXP en nous appuyant sur les résultats de l'étude de la section 8.2 du Chapitre 8 pour la modélisation des transmissions *broadcast*. Ensuite nous vérifions le modèle de RTXP en trois étapes :

1. Vérification temporelle ;
2. Ajout de la prise en compte des liens radios non-fiables ;
3. Vérification de grands réseaux.

Toutes ces étapes permettent de corriger la conception du protocole pour obtenir des garanties de respect des bornes temporelles fiables. La vérification temporelle permet de vérifier le protocole dans des cas simples pour corriger les erreurs de conception/modélisation les plus "grossières". La vérification incluant la prise en compte des liens non-fiables permet de trouver des topologies pour lesquels le protocole ne remplit pas sa mission et leurs probabilités d'occurrence. La vérification de grands réseaux permet de vérifier le passage à l'échelle de protocole et que la topologie étudiée permet d'écouler correctement le trafic applicatif.

Les résultats de vérification montrent que RTXP remplit bien sa mission d'acheminement de données en temps borné dans les RCsF, car aucune erreur de conception n'a été détectée. En effet, la seule erreur détecté est un dépassement de capacité (définie par l'Equation 5.7 p.94) lors de la vérification avec la méthode hybride dans la section précédente. Par ailleurs, dans ce chapitre, nous montrons que la méthodologie est applicable à un protocole temps-réel pour RCsF localisé qui est donc distribué.

Chapitre 10

Conclusion et perspectives

Cette thèse a pour objet les communications temps-réel dans les RCsF, nos travaux se sont articulés autour deux objectifs principaux : proposer des mécanismes protocolaires adaptés aux RCsF qui permettent d'assurer des transmissions temps-réel et proposer une méthodologie de vérification formelle de ces mécanismes.

Notre étude de l'état de l'art du Chapitre 2 montre que même si la QoS, et donc les contraintes temporelles, sont une préoccupation de plus en plus présente dans la conception de protocoles de RCsF, il manque des protocoles qui permettent à la fois de garantir un délai de bout en bout borné pour l'acheminement de données et de prendre en compte les aspects spécifiques des RCsF (énergie, large échelle, capacité). De tels protocoles sont nécessaires pour acheminer les données des applications critiques temps-réel qui émergent dans de nombreux contextes [Zhang et al., 2007] [Tan et al., 2010] [Ramesh, 2009]. La première partie de ce document permet, au travers de nos contributions, de combler ce manque. Nous mettons ensuite en évidence, dans le Chapitre 7, le peu de cas d'applications de méthodes formelles aux protocoles de RCsF. Les méthodes formelles sont pourtant un outil essentiel pour prouver que les protocoles supportent effectivement les exigences des applications critiques temps-réel [Clarke and Wing, 1996]. Nous proposons donc, dans la seconde partie de ce document, une méthodologie de vérification formelle qui répond aux problèmes posés par les spécificités des RCsF (énergie, large échelle, capacité). Dans les sections suivantes, nous résumons les contributions apportées dans ce document et les perspectives qu'ouvre ce travail.

10.1 Conclusions sur nos contributions

Nos contributions sont donc de deux types : des mécanismes protocolaires temps-réel pour les RCsF et une méthodologie de vérification formelle de ces mécanismes.

10.1.1 Contributions sur les mécanismes protocolaires

Dans le Chapitre 2, nous montrons que les seules solutions protocolaires qui permettent de garantir des délais de bout en bout bornés dans les RCsF ne tiennent pas

compte de toutes les spécificités des RCsF : notamment, les solutions d'ordonnancement global [Ergen and Varaiya, 2006] [Doherty and Pister, 2008] ne passent pas l'échelle du fait que l'ordonnancement est produit de manière centralisé. Nous déduisons de l'étude de la littérature les points qu'un protocole d'acheminement de données temps-réel pour RCsF doit satisfaire :

- permettre un accès au médium en temps borné ;
- borner le nombre de sauts nécessaires pour atteindre le puits ;
- passer l'échelle : utiliser des mécanismes localisés ;
- économiser l'énergie : permettre aux nœuds de dormir s'il n'y a pas de trafic à écouler ;
- être fiable : avoir une probabilité de succès de transmission de bout en bout suffisamment élevée pour satisfaire les besoins des applications critiques ;

Les contributions que nous proposons dans la première partie de ce document permettent de répondre à ces points :

1. le Chapitre 3 propose une étude de fiabilité des RCsF. La plupart des protocoles temps-réel proposés dans la littérature ne tiennent pas compte de la non-fiabilité du lien radio dans les RCsF, or les applications critiques nécessitent un niveau de fiabilité élevé (des vies humaines peuvent être en jeu). Nous proposons donc une étude de la fiabilité des routages classiques et opportunistes ; cette étude nous permet de conclure que le routage opportuniste devrait être préféré dans le contexte des systèmes critiques, car il fournit un niveau de fiabilité plus élevé que le routage classique. Or nous constatons que ce n'est pas le cas dans la littérature.
2. Dans le Chapitre 4, nous proposons un système de coordonnées virtuelles qui permet d'organiser le réseau pour introduire des mécanismes déterministes et acheminer les données vers le puits. Ce système de coordonnées à une dimension peut être vu comme un affinement du gradient : le nœud obtient sa coordonnée en fonction du nombre de sauts qui le sépare du puits et d'un paramètre basé sur le gradient des voisins du nœud qui affine cette valeur. Cette coordonnée permet également de discriminer avec une forte probabilité les nœuds dans le même domaine d'interférence (nous considérons un 2-voisinage). Nous proposons également un protocole de mise à jour de cette coordonnée, qui effectue la mise à jour seulement quand et où cela est nécessaire, c'est-à-dire quand du trafic circule et sur les parties du réseau empruntées par le trafic. Ce système de coordonnées est utilisé par les contributions suivantes.
3. Dans le Chapitre 5, nous proposons RTXP, un protocole *cross-layer* d'acheminement de données temps-réel pour RCsF qui répond à tous les points évoqués ci-dessus : il utilise le système de coordonnées virtuelles développé dans le chapitre 4 pour accéder au médium de manière déterministe et borner le nombre de sauts pour atteindre le puits depuis n'importe quelle source. L'utilisation de la coordonnée permet, de plus, de n'utiliser qu'une information locale pour prendre les décisions d'accès au canal et de routage. La durée d'éveil des nœuds dépend du trafic : des périodes d'activité secondaires sont déclenchées s'il y a du trafic à écouler. L'utilisation d'un schéma de routage opportuniste permet

une meilleure fiabilité des transmissions de bout en bout par rapport aux protocoles auxquels on compare RTXP. Même s'il n'est pas possible de garantir absolument un délai de bout en bout borné sur des liens non-fiables, nous montrons tout de même que les mécanismes déterministes de RTXP, qui permettent d'éviter les collisions, induisent un taux de réception de paquets avant échéance plus important que pour une solution avec des accès aléatoires.

4. Dans le Chapitre 6, nous nous intéressons au problème de tempête d'alarmes : lors de l'apparition d'un événement, il est possible que de nombreux nœuds détectent le phénomène au même instant et envoient de nombreux paquets au puits, contenant des informations redondantes. Cela induit une congestion du réseau et donc des collisions pour l'accès au médium et une surconsommation d'énergie. Nous proposons R^2A, un protocole d'agrégation qui permet de sélectionner un seul émetteur de l'alarme de détection (ou un petit nombre : les résultats de simulation montrent que dans la plupart des cas il y a moins de 3 émetteurs). Le mécanisme qui permet à R^2A de sélectionner le nœud émetteur est basé sur une compétition dont l'issue dépend des valeurs des coordonnées (présentées dans le Chapitre 4) des nœuds participants. R^2A s'exécute seulement sur détection d'un événement et seulement pendant les périodes de sommeil de RTXP, cela permet de ne pas introduire de délais supplémentaires dans le délai de bout en bout.

Ces travaux sont une première contribution de protocoles temps-réel fiables adaptés à des réseaux radios multi-sauts large échelle contraints en énergie qui permettent de supporter les applications critiques de détection d'anomalies. Dans la section 10.2 nous présentons les perspectives qu'ouvrent ces travaux.

10.1.2 Contributions sur la vérification formelle

Dans la seconde partie de ce document, nous commençons, dans le Chapitre 7, par faire un état de l'art des méthodes formelles de vérification. Nous nous attachons à sélectionner une méthode, un formalisme de modélisation et un outil de vérification qui permettent de modéliser et vérifier tous les aspects des RCsF :

- la concurrence : chaque nœud exécute des actions en parallèle des autres nœuds ;
- les transmissions : les nœuds voisins dans la topologie peuvent communiquer entre eux et interfèrent entre eux du fait de la nature *broadcast* du médium radio ;
- les données : il faut pouvoir disposer de structures de données pour représenter les informations du système (informations contenues dans les paquets, topologie du réseau, etc) ;
- les liens radios non-fiables, il faut pouvoir représenter les probabilités de pertes de paquets ;
- le temps : dans ce document, nous nous intéressons aux comportements temporels des protocoles pour RCsF.

Nous montrons que peu d'outils permettent la modélisation et la vérification de tous ces aspects et que le *model checker* UPPAAL [Gerd Behrmann and Larsen, 2004]

semble le plus approprié dans ce contexte. Cependant, trois points sont essentiels pour la modélisation et vérification formelle des RCsF et sont peu étudiés dans la littérature :

- la modélisation de la nature *broadcast* des transmissions sans fil ;
- la non-fiabilité du lien radio ;
- le passage à l'échelle nécessaire pour vérifier des réseaux de grandes tailles (plusieurs centaines de nœuds).

Les contributions que nous proposons dans la seconde partie de ce document permettent de répondre à ces trois points, elles sont présentées dans le Chapitre 8 :

1. Nous effectuons une étude des méthodes de modélisation des transmissions *broadcast* : il existe dans la littérature trois façons de modéliser ce comportement, nous comparons les performances de ces méthodes lors de la vérification par *Model Checking*. Nous mettons en évidence que la méthode de vérification de la connectivité avant synchronisation permet d'atténuer l'explosion combinatoire de l'espace de configurations et doit donc être préférée pour la vérification formelle des RCsF qui sont large échelle. Cette étude nous donne une bonne pratique pour modéliser cet aspect des réseaux radio et permet donc de produire des modèles de protocoles. Cependant, les méthodes proposées ne prennent pas en compte la non-fiabilité des liens radio.
2. Nous proposons une méthode de vérification qui permet de prendre en compte les liens radios non-fiables. Cette méthode consiste à vérifier les protocoles sur toutes les topologies logiques possibles à partir de la position géographique des nœuds. Chaque topologie logique a une probabilité d'existence qui dépend de la probabilité de succès de transmissions sur les liens radio. La méthode permet donc de déterminer sur quelles topologies logiques le protocole fonctionne bien (la propriété est vérifiée) et donc quelle est la probabilité de bon fonctionnement du protocole. Cependant, cette méthode permet de vérifier seulement des réseaux de taille modeste, nous proposons donc une méthode pour passer l'échelle.
3. Nous proposons une méthode hybride qui allie *Network Calculus* et *Model Checking* : elle tire parti de la capacité de passage à l'échelle du *Network Calculus* et de l'exhaustivité du *Model Checking*. Cette méthode consiste à vérifier, par *Model Checking*, que l'hypothèse de trafic que constitue la courbe de service de *Network Calculus* est vraie pour tous les nœuds du réseau. La complexité algorithmique de cette méthode croît avec les bornes sur le trafic (courbes d'arrivées), cela permet de vérifier des réseaux de grandes tailles (dans le Chapitre 9 nous vérifions RTXP sur des réseaux de 500 nœuds).

Nous articulons ces trois propositions pour former une méthodologie de vérification formelle des protocoles temps-réel pour RCsF. Cette méthodologie est composée d'une étape de modélisation et de trois étapes de vérification : l'étape de modélisation consiste à produire le modèle du protocole à vérifier en utilisant notamment la méthode de modélisation des transmissions *broadcast* préconisée par l'étude proposée précédemment (proposition 1 de la seconde partie de ce document). La première étape de vérification consiste à vérifier le protocole sans prendre en compte les liens non-

fiables pour détecter les erreurs qui apparaissent dans ce cas ; la deuxième étape de vérification consiste à vérifier le protocole en prenant en compte les liens radios non-fiable pour détecter les erreurs qui se produisent avec différentes topologies logiques et estimer la probabilité d'apparition de ces mauvais comportements ; la troisième étape de vérification consiste à appliquer la méthode de vérification hybride pour vérifier que le protocole étudié passe l'échelle. Nous appliquons cette méthodologie à RTXP dans le Chapitre 9 et concluons que, pour les cas considérés dans ce chapitre, RTXP respecte ses spécifications, à savoir : il est capable d'acheminer des alarmes vers le puits en temps borné.

La méthodologie que nous proposons est à notre connaissance la première tentative de formalisation des éléments et des étapes nécessaires à la vérification formelle des protocoles temps-réel pour RCsF. Dans la section suivante, nous présentons des pistes qui mériteront d'être explorées pour améliorer nos propositions.

10.2 Perspectives

Les communications temps-réel dans les RCsF sont un domaine encore peu étudié [Decotignie, 2012], les contributions de cette thèse explorent une partie des problèmes que posent ce domaine. Cependant, de nombreux problèmes de ce domaine restent ouverts. Dans cette section nous présentons perspectives qu'ouvrent nos travaux aux niveaux protocolaire et vérification formelle.

10.2.1 Sur les mécanismes protocolaires

Dans ce document, nous proposons des mécanismes protocolaires pour acheminer des données en temps-réel dans les RCsF. Nous évaluons ces propositions de manière théorique, par simulation et vérifions formellement leurs propriétés. Ces étapes d'évaluation sont nécessaires, mais pour montrer la faisabilité de notre approche, il faut également l'implémenter sur les nœuds capteurs et la tester dans le cas de scénarios réalistes. Nous prévoyons de réaliser cela dans le futur. On note tout de même que des protocoles mettant en jeux des mécanismes similaires à notre solution ont déjà été implémentés avec succès [Watteyne et al., 2009b] [Boano et al., 2012] [Decotignie, 2012].

En l'état, RTXP est une solution de remontée d'alarmes, or il serait intéressant de pouvoir réaliser non seulement de la détection d'événement, mais aussi de l'observation en temps-réel du phénomène détecté. Par exemple, dans le cas de l'application de détection de feux de forêt, il est important que l'alarme qui signale un départ de feu soit effectivement reçue dans un temps borné, mais il est ensuite très intéressant pour les pompiers de connaître la propagation du feu en temps-réel pour appliquer une stratégie de lutte plus efficace. Une piste pour mettre en place un protocole permettant à la fois la détection et l'observation serait de fonctionner en deux modes : un mode de détection dans lequel les nœuds dorment la plupart du temps pour économiser l'énergie (ils doivent tout de même se réveiller assez souvent pour pouvoir respecter l'échéance de détection de l'événement) et un mode d'observation durant

lequel les nœuds ne dorment quasiment pas car l'observation de l'événement est plus importante que l'économie d'énergie (d'ailleurs dans le cas du feu de forêt, les capteurs sont détruits par le feu à terme, il est donc inutile d'économiser l'énergie). Ce type de protocoles a déjà été proposé dans la littérature ([Chen et al., 2007] par exemple) mais sans garantie sur les délais de bout en bout.

Enfin, une autre piste à explorer est celle de la remontée de plusieurs types d'alarmes avec des contraintes temporelles variées. En effet, à terme pour des raisons économiques (coûts de déploiement/maintenance), les RCsF devront supporter plusieurs applications critiques qui s'exécuteront en parallèle avec des contraintes temporelles différentes. Les travaux en *mixed criticality* [Baruah et al., 2011] sur l'ordonnancement temps-réel commencent à aborder ce sujet sur des systèmes non distribués : la *mixed criticality* consiste étudier et certifier des systèmes sur lesquels s'exécutent à la fois des applications critiques et non-critiques [Baruah et al., 2011]. Il serait intéressant d'appliquer les méthodes de vérification proposées dans ce contexte aux réseaux et en particulier aux RCsF.

10.2.2 Sur la vérification formelle

Pour la vérification formelle, notre méthodologie permet de vérifier que des protocoles de RCsF sont capables de respecter des échéances temporelles en trois étapes : d'abord sans lien non-fiable et sur de petits réseaux pour détecter les erreurs dans les cas les plus simples, puis avec les liens non-fiables toujours dans des réseaux de quelques nœuds et enfin dans des réseaux large échelle mais sans la prise en compte des liens non-fiables. Ces étapes de vérification permettent de donner des garanties quant au bon fonctionnement d'un protocole temps-réel pour RCsF. Cependant, il serait intéressant de pouvoir vérifier des systèmes larges échelles et probabilistes à la fois. Une première approche pour résoudre ce problème serait l'utilisation de versions probabiliste de *Network Calculus* [Jiang, 2006] [Santinelli and Cucu-Grosjean, 2011] dans notre méthode hybride présentée dans le Chapitre 8. Cependant, ces approches permettent seulement d'associer des distributions de probabilités aux courbes de service et d'arrivées, la topologie du réseau, elle, ne change pas. Dans le cas de RCsF, la topologie du réseau est dynamique et donc le routage l'est aussi, à notre connaissance cet aspect n'est pas représentable en *Network Calculus*.

Comme nous le décrivons au fil de ce document, il n'est pas possible de donner de garanties absolues sur des systèmes réels, car ces systèmes sont soumis à un environnement physique probabiliste. Il est possible que des paquets soient perdus ou arrivent après l'échéance, il faut donc être capable de comprendre et gérer ce risque. Dans le Chapitre 3, nous évaluons théoriquement la fiabilité des transmissions de bout en bout, puis dans le Chapitre 5 nous l'évaluons par simulation et enfin dans le Chapitre 8, nous proposons une méthode pour évaluer la probabilité des cas où le protocole a un bon fonctionnement. Toutes ces estimations de la fiabilité du système sont basées sur un modèle théorique de propagation (dans ce document nous utilisons le modèle *log-normal shadowing*), or ce modèle ne capture pas tous les phénomènes rencontrés dans la réalité (notamment les liens asymétriques, la corrélation spatiale, etc). Une piste à explorer pour lever ces limitations serait l'usage d'expérimentations réelles et

l'application de la théorie des valeurs extrêmes [Wikipedia, 2013a] sur les résultats d'expérimentation. La théorie des valeurs extrêmes est une branche des statistiques qui s'intéresse à la probabilité d'événements extrêmes comme les grandes crues de fleuves ou bien les tremblements de terre. Cette théorie est également utilisée pour estimer des distributions de probabilités des pires cas d'exécutions de tâche en théorie de l'ordonnancement [Bernat et al., 2003]. En appliquant les outils de cette théorie sur des données de délais de bout en bout dans des RCsF réels, il serait possible d'estimer la probabilité d'apparition de délais qui dévient de manière extrême de la moyenne et ainsi estimer la probabilité de dépasser l'échéance donnée par l'application.

Pour conclure, les perspectives offertes par cette thèse, comme nos propositions, sont variées et touchent de nombreux domaines. Notre travail est un premier pas dans ces domaines et devrait permettre à terme d'évoluer vers des systèmes informatiques ubiquitaire temps-réel et sûrs.

Annexe A
Déclarations du modèle UPPAAL de RTXP

Cette annexe présente les déclarations de variables et fonctions (exécutées lors des passages de transitions) associées au TA UPPAAL de la Figure 9.5 p.208.

A.1 Déclarations globales

La portée des variables déclarées ci-dessous est globale : leurs valeurs sont communes à tous les nœuds.

```
 // Global declarations.
broadcast chan c;
bool ok=0;

const int N=;//place number of nodes here
bool jammingTab[N];
int[0,N] msgTab[N];
const int DC=99;
int[0,N-1] sr=0;
clock a;

const int rcvSlot=160;
const int backoffD=102;
const int jamming=2;
const int M=3;


 const bool connect1[N][N]=//place topology here
/*
example:

const bool connect1[N][N]={{0,1,1,1,1},
{1,0,0,0,1},
```

```
{1,0,0,0,0},
{1,0,0,0,1},
{1,1,0,1,0}};

end example
*/

const bool connect2[N][N]=//place 2-hop topology here
```

A.2 Déclarations locales à un nœud

La portée des variables déclarées ci-dessous est locale : leurs valeurs sont différentes pour chaque nœud. Une configuration du NTA contient donc une instance de ces variables par nœud.

```
 // Local declarations.

bool L=0;
bool LHeard=0;
clock x,y,z;
int[0,N] m[M]={id,0,0};
int[0,N] currentMsg=0;



bool connectedJ(){
    int i;
    for(i=0;i<N;i++){
        if(connect2[id][i] && jammingTab[i]){
            return true;
        }
    }
    return false;
}

void sendJ(){
    jammingTab[id]=1;
}

void resetJ(){
    jammingTab[id]=0;
}
```

```
void sendP(){
    int i;
    currentMsg=m[0];
    for(i=0;i<M-1;i++){
        m[i]=m[i+1];
    }
    m[M-1]=0;
    msgTab[id]=currentMsg;
}

bool connected1(){
    int i;

    for(i=0;i<N;i++){
        if(connect1[id][i] && msgTab[i]!=0){
    return true;
        }
    }
    return false;
}

bool connected2(){
    int i;
    int nb=0;
    for(i=0;i<N;i++){
        if(connect2[id][i] && msgTab[i]!=0){
        nb=nb+1;
        }
    }
    if(nb<=1){
        return false;
    }else{
        return true;
    }
}

void upMsg(){
    int i;

    for(i=0;i<N;i++){
        if(connect1[id][i] && msgTab[i]!=0){
            currentMsg=msgTab[i];
    if(id==0){
sr=sr+1;
ok=1;
```

```
    }
        }
    }
}

void resetP(){
    msgTab[id]=0;
}

void accept(){
    int i;
    bool found=0;

    for(i=0;i<3;i++){
        if(!found && m[i]==0){
            m[i]=currentMsg;
            found=1;
        }
    }
}

void upmts(){
    int i;
    mts=0;

    for(i=0;i<M;i++){
        if(m[i]!=0){
            mts=1;
        }
    }
    if(id==0){
        mts=0;
    }
}

void Nok(){
    if(id==0){
ok=0;
    }
}

void mmsg(){
    int i;
    int n;
    for(i=0;i<M;i++){
```

```
        if(m[i]!=0){
            n=n+1;
        }
    }
    if(n>1 && !LHeard && !L && id!=0){
L=1;
    }
}
```

Bibliographie

[Abadi and Fournet, 2001] Abadi, M. and Fournet, C. (2001). Mobile values, new names, and secure communication. In *ACM SIGPLAN Notices*, volume 36, pages 104–115. ACM.

[Abichar and Chang, 2005] Abichar, Z. G. and Chang, J. M. (2005). Conti : constant-time contention resolution for wlan access. IFIP NETWORKING, pages 358–369, Waterloo, Canada.

[AboElFotoh et al., 2006] AboElFotoh, H. M., ElMallah, E. S., and Hassanein, H. S. (2006). On the reliability of wireless sensor networks. IEEE ICC, pages 3455–3460, Istanbul, Turkey.

[Al-Karaki and Kamal, 2004] Al-Karaki, J. N. and Kamal, A. E. (2004). Routing techniques in wireless sensor networks : a survey. *Wireless Communications, IEEE*, 11(6) :6–28.

[Al-Karaki et al., 2009] Al-Karaki, J. N., Ul-Mustafa, R., and Kamal, A. E. (2009). Data aggregation and routing in wireless sensor networks : Optimal and heuristic algorithms. *Computer networks*, 53(7) :945–960.

[Alphasense-B4, 2013] Alphasense-B4 (2013). http ://www.alphasense.com/industrial-sensors/pdf/no2b4.pdf, viewed 8 september 2013.

[Altisen et al., 2012] Altisen, K., Devismes, S., Gerbaud, A., and Lafourcade, P. (2012). Analysis of random walks using tabu lists. In *Structural Information and Communication Complexity*, pages 254–266. Springer.

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

[Amadou, 2012] Amadou, I. (2012). Protocoles de routage sans connaissance du voisinage pour réseaux radio multi-sauts. Phd thesis manuscript, insa de lyon.

[ARESA2, 2013] ARESA2 (2013). http ://aresa2.orange-labs.fr/, viewed 7 september 2013.

[Atamna, 1994] Atamna, Y. (1994). *Réseaux de Petri temporisés stochastiques classiques et bien formés : définition, analyse et application aux systèmes distribués temps réel*. PhD thesis.

[Augé-Blum et al., 2012] Augé-Blum, I., Boussetta, K., Rivano, H., Stanica, R., and Valois, F. (2012). Capillary networks : a novel networking paradigm for urban environments. In *Proceedings of the first workshop on Urban networking*, pages 25–30. ACM.

[Augé-Blum et al., 2011] Augé-Blum, I., Yang, F., and Watteyne, T. (2011). Real-Time Communications in Wireless Sensor Networks. *Next Generation Mobile Networks and Ubiquitous Computing, Samuel Pierre Ed., Information Science Reference (an imprint of IGI Global), ISBN 978-1-60566-250-3.*

[Bachir et al., 2010] Bachir, A., Dohler, M., Watteyne, T., and Leung, K. K. (2010). Mac essentials for wireless sensor networks. *Communications Surveys & Tutorials, IEEE*, 12(2) :222–248.

[Baeten, 2005] Baeten, J. (2005). A brief history of process algebra. *Theoretical Computer Science*, 335(2-3) :131–146.

[Baier and Katoen, 2008] Baier, C. and Katoen, J.-P. (2008). *Principles of Model Checking*. The MIT Press.

[Bao and Garcia-Luna-Aceves, 2001] Bao, L. and Garcia-Luna-Aceves, J. (2001). A new approach to channel access scheduling for ad hoc networks. ACM MobiCom, pages 210–221, Rome, Italy.

[Baruah et al., 2011] Baruah, S. K., Burns, A., and Davis, R. I. (2011). Response-time analysis for mixed criticality systems. IEEE RTSS, pages 34–43, Vienna, Austria.

[Bauer et al., 2010] Bauer, H., Scharbarg, J.-L., and Fraboul, C. (2010). Worst-case end-to-end delay analysis of an avionics afdx network. DATE, pages 1220–1224, Dresden, Germany.

[Berendsen et al., 2009] Berendsen, J., Chen, T., and Jansen, D. (2009). Undecidability of cost-bounded reachability in priced probabilistic timed automata. *Theory and Applications of Models of Computation*, 214755 :128–137.

[Berendsen et al., 2010] Berendsen, J., Jansen, D., and Vaandrager, F. (2010). Fortuna : Model checking priced probabilistic timed automata. QEST, pages 273–281, Williamsburg, USA.

[Bergstra and Klop, 1984] Bergstra, J. A. and Klop, J. W. (1984). Process algebra for synchronous communication. *Information and Control*, pages 109–137.

[Bernat et al., 2003] Bernat, G., Colin, A., and Petters, S. (2003). pwcet : A tool for probabilistic worst-case execution time analysis of real-time systems. *REPORT-UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE YCS.*

[Biswas and Morris, 2005] Biswas, S. and Morris, R. (2005). Exor : opportunistic multi-hop routing for wireless networks. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 133–144. ACM.

[Boano et al., 2012] Boano, C. A., Zúniga, M. A., Romer, K., and Voigt, T. (2012). Jag : Reliable and predictable wireless agreement under external radio interference. IEEE RTSS, pages 315–326. IEEE.

[Bozga et al., 2004] Bozga, M., Graf, S., Ober, I., Ober, I., and Sifakis, J. (2004). The if toolset. In *Formal Methods for the Design of Real-Time Systems*, pages 237–267. Springer.

[Bron and Kerbosch, 1973] Bron, C. and Kerbosch, J. (1973). Algorithm 457 : finding all cliques of an undirected graph. *Commun. ACM*, 16(9) :575–577.

[Buettner et al., 2006] Buettner, M., Yee, G. V., Anderson, E., and Han, R. (2006). X-mac : A short preamble mac protocol for duty-cycled wireless sensor networks. ACM SenSys, pages 307–320, Boulder, USA.

[Caccamo et al., 2002] Caccamo, M., Zhang, L. Y., Sha, L., and Buttazzo, G. (2002). An implicit prioritized access protocol for wireless sensor networks. IEEE RTSS, pages 39–48, Austin, USA.

[Caleffi et al., 2008] Caleffi, M., Ferraiuolo, G., and Paura, L. (2008). A reliability-based framework for multi-path routing analysis in mobile ad-hoc networks. *International Journal of Communication Networks and Distributed Systems*, 1(4) :507–523.

[Cao and Abdelzaher, 2004] Cao, Q. and Abdelzaher, T. (2004). A scalable logical coordinates framework for routing in wireless sensor networks. IEEE RTSS, pages 349–358, Lisbon, Portugal.

[Chao and Hsiao, 2013] Chao, C.-M. and Hsiao, T.-Y. (2013). Design of structure-free and energy-balanced data aggregation in wireless sensor networks. *Journal of Network and Computer Applications*.

[Chen and Varshney, 2004] Chen, D. and Varshney, P. K. (2004). Qos support in wireless sensor networks : A survey. volume 233 of *ICWN*, Las Vegas, USA.

[Chen et al., 2007] Chen, J., Zhu, P., and Qi, Z. (2007). Pr-mac : Path-oriented real-time mac protocol for wireless sensor network. IEEE ICESS '07, pages 530–539, Daegu, Korea.

[Chen et al., 2004] Chen, Y., Farley, T., and Ye, N. (2004). Qos requirements of network applications on the internet. *Information, Knowledge, Systems Management*, 4(1) :55–76.

[Chipara et al., 2006] Chipara, O., He, Z., Xing, G., Chen, Q., Wang, X., Lu, C., Stankovic, J., and Abdelzaher, T. (2006). Real-time power-aware routing in sensor networks. IEEE IWQoS, pages 83–92.

[Clark et al., 1990] Clark, B. N., Colbourn, C. J., and Johnson, D. S. (1990). Unit disk graphs. *Discrete mathematics*, 86(1) :165–177.

[Clarke and Wing, 1996] Clarke, E. and Wing, J. (1996). Formal methods : State of the art and future directions. *ACM Computing Surveys*, 28(4) :626–643.

[Cleaveland and Sims, 2002] Cleaveland, R. and Sims, S. T. (2002). Generic tools for verifying concurrent systems. *Science of Computer Programming*, 42(1) :39–47.

[Coleri, 2002] Coleri, S. (2002). Pedamacs : Power efficient and delay aware medium access protocol for sensor networks, ms thesis, electrical engineering and computer science, university of california, berkeley.

[Coq, 2010] Coq (2010). The coq proof assistant reference manual version 8.3. INRIA.

[Coronis, 2013] Coronis (2013). http ://www.coronis.com/en/wavesense.html, viewed 6 september 2013.

[Davis et al., 2007] Davis, R. I., Burns, A., Bril, R. J., and Lukkien, J. J. (2007). Controller area network (can) schedulability analysis : Refuted, revisited and revised. *Real-Time Systems*, 35(3) :239–272.

[Decotignie, 2012] Decotignie, J. (2012). Real-time and wireless sensor networks : Are they compatible. In *Keynote at Euromicro Conference on Real-Time Systems*.

[Doherty and Pister, 2008] Doherty, L. and Pister, K. S. J. (2008). Tsmp : Time synchronized mesh protocol. IEEE/IFIP DSN, pages 391–398, Orlando, USA.

[Dohler et al., 2007] Dohler, M., Barthel, D., Maraninchi, F., Mounier, L., Aubert, S., Dugas, C., Buhrig, A., Paugnat, F., Renaudin, M., Duda, A., Heusse, M., and Valois, F. (2007). The aresa project : Facilitating research, development and commercialization of wsns. IEEE SECON, pages 590–599.

[Dunkels, 2011] Dunkels, A. (2011). The contikimac radio duty cycling protocol. Technical report t2011 :13.

[Ellsberger et al., 1997] Ellsberger, J., Hogrefe, D., and Sarma, A. (1997). *SDL : formal object-oriented language for communicating systems*. Prentice Hall.

[Elson and Estrin, 2003] Elson, J. E. and Estrin, D. (2003). *Time synchronization in wireless sensor networks*. PhD thesis, University of California, Los Angeles.

[Engdahl and Haugstad, 2007] Engdahl, R. and Haugstad, A. (2007). *State space reduction for probabilistic timed automata*. PhD thesis, Aalborg University.

[Engdahl and Haugstad, 2008] Engdahl, R. and Haugstad, A. (2008). *Efficient Model Checking for Probabilistic Timed Automata*. PhD thesis, Aalborg University.

[Enz et al., 2004] Enz, C. C., El-Hoiydi, A., Decotignie, J.-D., and Peiris, V. (2004). Wisenet : an ultralow-power wireless sensor network solution. *Computer*, 37(8) :62–70.

[Erdene-Ochir et al., 2010] Erdene-Ochir, O., Minier, M., Valois, F., and Kountouris, A. (2010). Resiliency of wireless sensor networks : definitions and analyses. In *IEEE ICT*, pages 828–835, Doha, Qatar.

[Ergen and Varaiya, 2006] Ergen, S. and Varaiya, P. (2006). Pedamacs : power efficient and delay aware medium access protocol for sensor networks. In *IEEE Transactions on Mobile Computing*, volume 5, pages 920–930.

[ETSI, 2011] ETSI (2011). Electromagnetic compatibility and Radio spectrum Matters (ERM) ; System Reference document (SRdoc) : Spectrum Requirements for Short Range Device, Metropolitan Mesh Machine Networks (M3N) and Smart Metering (SM) applications. Technical Report TR 103 055 V1.1.1, ETSI.

[Fan et al., 2006] Fan, K.-W., Liu, S., and Sinha, P. (2006). Scalable data aggregation for dynamic events in sensor networks. Sensys, pages 181–194, Boulder, USA.

[Fehnker et al., 2007] Fehnker, A., Van Hoesel, L., and Mader, A. (2007). Modelling and verification of the lmac protocol for wireless sensor networks. IFM, pages 253–272, Oxford, UK.

[Felemban et al., 2006] Felemban, E., Lee, C., and Ekici, E. (2006). Mmspeed : Multipath multi-speed protocol for qos guarantee of reliability and timeliness in wireless sensor networks. In *IEEE Transactions on Mobile Computing*, volume 5, pages 738–754.

[Ferrand et al., 2013] Ferrand, P., Gorce, J.-M., and Goursaud, C. (2013). Approximations of the packet error rate under slow fading in direct and relayed links. Rapport de recherche RR-8316, INRIA.

[Ferreira et al., 2004] Ferreira, J., Oliveira, A., Fonseca, P., and Fonseca, J. (2004). An experiment to assess bit error rate in can. RTN, pages 15–18, Catania, Italy.

[Fraboulet et al., 2007] Fraboulet, A., Chelius, G., and Fleury, E. (2007). Worldsens : development and prototyping tools for application specific wireless sensors networks. IPSN, pages 176–185, Cambridge, USA.

[FreeRTOS, 2013] FreeRTOS (2013). http ://www.freertos.org/, viewed 8 september 2013.

[Fruth, 2006] Fruth, M. (2006). Probabilistic model checking of contention resolution in the ieee 802.15. 4 low-rate wireless personal area network protocol. In *IEEE ISoLA*, pages 290–297, Paphos, Cyprus.

[Fruth, 2011] Fruth, M. (2011). *Formal Methods for the Analysis of Wireless Network Protocols*. PhD thesis, Oxford University.

[Garavel et al., 2013] Garavel, H., Lang, F., Mateescu, R., and Serwe, W. (2013). Cadp 2011 : a toolbox for the construction and analysis of distributed processes. *International Journal on Software Tools for Technology Transfer*, pages 1–19.

[Geilen, 2002] Geilen, M. (2002). Formal techniques for verification of complex real-time systems. *PhD thesis, Eindhoven University of Technology*.

[Gerd Behrmann and Larsen, 2004] Gerd Behrmann, A. D. and Larsen, K. (2004). A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*, pages 33–35.

[Gnawali et al., 2008] Gnawali, O., Fonseca, R., Jamieson, K., and Levis, P. (2008). Ctp : Robust and efficient collection through control and data plane integration. Technical report, Technical report, Univ. of Southern California, UC Berkeley, MIT CSAIL, Stanford Univ.

[Godary, 2004] Godary, K. (2004). Validation temporelle de r'eseaux embarqu'es critiques et fiables pour l'automobile. Phd thesis manuscript, insa lyon, universit'e de lyon, to appear.

[Godskesen and Gryn, 2006] Godskesen, J. C. and Gryn, O. (2006). Modelling and verification of security protocols for ad hoc networks using uppaal. 18th Nordic Workshop on Programming Theory.

[Goldsmith, 2005] Goldsmith, A. (2005). *Wireless Communications*. Cambridge University Press.

[Gupta and Kumar, 2000] Gupta, P. and Kumar, P. (2000). The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2) :388–404.

[Haddad, 2011] Haddad, S. (2011). Process algebra for synchronous communication. *actes de l'Ecole D"et'e Temps-R'eel*, pages 49–56.

[Hansson and Jonsson, 1990] Hansson, H. and Jonsson, B. (1990). A calculus for communicating systems with time and probabilities. IEEE RTSS, pages 278–287, Lake Buena Vista, USA.

[Heinzelman et al., 2000] Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. IEEE HICSS.

[Henzinger et al., 1992] Henzinger, T., Nicollin, X., Sifakis, J., and Yovine, S. (1992). Symbolic model checking for real-time systems. IEEE LICS, pages 394–406, Santa Cruz, USA.

[Heurtefeux et al., 2012] Heurtefeux, K., Fabrice, V., et al. (2012). De la pertinence du rssi pour la localisation dans les réseaux de capteurs. *Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*.

[Hoare, 1985] Hoare, C. A. R. (1985). *Communicating sequential processes*. Prentice-Hall, Inc.

[Huang et al., 2009] Huang, P., Chen, H., Xing, G., and Tan, Y. (2009). Sgf : A state-free gradient-based forwarding protocol for wireless sensor networks. In *ACM Trans. Sen. Netw.*, volume 5, pages 14 :1–14 :25.

[IEEE, 2003] IEEE (2003). Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. part 11 : Wireless lan medium access control (mac) and physical layer (phy) specifications. amendment 4 : Further higher data rate extension in the 2.4 ghz band.

[IEEE, 2011a] IEEE (2011a). Ieee guide–adoption of the project management institute (pmi(r)) standard a guide to the project management body of knowledge (pmbok(r) guide)–fourth edition. *IEEE Std 1490-2011*, pages 1–508.

[IEEE, 2011b] IEEE (2011b). Ieee standard for local and metropolitan area networks - part 15.4 : Low-rate wireless personal area networks (lr-wpans).

[IEEE, 2012] IEEE (2012). IEEE Standard for Local and metropolitan area networks - Part 15.6 : Wireless Body Area Networks.

[IETF, 2013] IETF (2013). The lightweight on-demand ad hoc distance-vector routing protocol - next generation (loadng).

[IETF-ROLL, 2011] IETF-ROLL (2011). Rpl : Ipv6 routing protocol for low-power and lossy networks.

[Intanagonwiwat et al., 2000] Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion : a scalable and robust communication paradigm for sensor networks. ACM MobiCom, pages 56–67, Boston, USA.

[Isabelle, 2011] Isabelle (2011). The isabelle/isar reference manual. University of Cambridge, Technische Universitaet Muenchen.

[Jiang, 2006] Jiang, Y. (2006). A basic stochastic network calculus. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 123–134. ACM.

[Jou and Smolka, 1990] Jou, C.-C. and Smolka, S. (1990). Equivalences, congruences, and complete axiomatizations for probabilistic processes. In *CONCUR '90 Theories of Concurrency : Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 367–383.

[Jouhet, 2012] Jouhet, L. (2012). Algorithmique du network calculus. Phd thesis manuscript, ens lyon, universit'e de lyon.

[Kai-Wei Fan et al., 2007] Kai-Wei Fan, K.-W., Liu, S., and Sinha, P. (2007). Structure-free data aggregation in sensor networks. *IEEE Transactions on Mobile Computing*, 6(8) :929–942.

[Karp and Kung, 2000] Karp, B. and Kung, H. T. (2000). Gpsr : greedy perimeter stateless routing for wireless networks. ACM MobiCom, pages 243–254, Boston, USA.

[Kopetz, 2011] Kopetz, H. (2011). *Real-time systems : design principles for distributed embedded applications*. Springer.

[Krishnamachari et al., 2002] Krishnamachari, L., Estrin, D., and Wicker, S. (2002). The impact of data aggregation in wireless sensor networks. IEEE ICDCS, pages 575–578, Vienna, Austria.

[Künzli et al., 2007] Künzli, S., Hamann, A., Ernst, R., and Thiele, L. (2007). Combined approach to system level performance analysis of embedded systems. IEEE/ACM CODES+ISSS, pages 63–68.

[Kwiatkowska et al., 2009] Kwiatkowska, M., Norman, G., and Parker, D. (2009). Stochastic games for verification of probabilistic timed automata. *Formal Modeling and Analysis of Timed Systems*, pages 212–227.

[Kwiatkowska et al., 2011] Kwiatkowska, M., Norman, G., and Parker, D. (2011). PRISM 4.0 : Verification of probabilistic real-time systems. CAV, pages 585–591, Snowbird, USA.

[Kwiatkowska et al., 2006] Kwiatkowska, M., Norman, G., Parker, D., and Sproston, J. (2006). Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29(1) :33–78.

[Kwiatkowska et al., 2002a] Kwiatkowska, M., Norman, G., Segala, R., and Sproston, J. (2002a). Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282(1) :101–150.

[Kwiatkowska et al., 2002b] Kwiatkowska, M., Norman, G., and Sproston, J. (2002b). Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. *Process Algebra and Probabilistic Methods : Performance Modeling and Verification*, pages 1–8.

[Lampin, 2013] Lampin, Q. (2013). Qos and time-constrained wireless sensor networks. Phd thesis manuscript, insa lyon, universit'e de lyon, to appear.

[Lampin et al., 2012] Lampin, Q., Barthel, D., Augé-Blum, I., and Valois, F. (2012). Qos oriented opportunistic routing protocol for wireless sensor networks. IEEE/IFIP WD, pages 1–6, Dublin, Ireland.

[Lampka et al., 2009] Lampka, K., Perathoner, S., and Thiele, L. (2009). Analytic real-time analysis and timed automata : a hybrid method for analyzing embedded real-time systems. ACM EMSOFT, pages 107–116, Grenoble, France.

[Lann and Rivière, 1993] Lann, G. L. and Rivière, N. (1993). Real-Time Communications over Broadcast Networks : the CSMA-DCR and the DOD-CSMA-CD Protocols. Research Report 1873, INRIA.

[Le Boudec and Thiran, 2001] Le Boudec, J.-Y. and Thiran, P. (2001). *Network calculus : a theory of deterministic queuing systems for the internet*, volume 2050. Springer.

[Liu and Layland, 1973] Liu, C. L. and Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1) :46–61.

[Liu et al., 2009] Liu, S., Fan, K.-W., and Sinha, P. (2009). Cmac : an energy-efficient mac layer protocol using convergent packet forwarding for wireless sensor networks. *ACM Transactions on Sensor Networks*, 5(4) :29.

[Loscri et al., 2005] Loscri, V., Morabito, G., and Marano, S. (2005). A two-levels hierarchy for low-energy adaptive clustering hierarchy (tl-leach). IEEE VTC, Dallas, USA.

[Lu et al., 2004] Lu, G., Krishnamachari, B., and Raghavendra, C. (2004). An adaptive energy-efficient and low-latency mac for data gathering in wireless sensor networks. IEEE IPDPS, Santa Fe, USA.

[Lu et al., 2010] Lu, J., Valois, F., Dohler, M., and Wu, M.-Y. (2010). Optimized data aggregation in wsns using adaptive arma. IARIA SENSORCOMM, pages 115–120.

[Lu et al., 2008] Lu, J.-L., Valois, F., Dohler, M., and Barthel, D. (2008). Quantifying organization by means of entropy. *Communications Letters, IEEE*, 12(3) :185–187.

[Macedo et al., 2009] Macedo, M., Grilo, A., and Nunes, M. (2009). Distributed latency-energy minimization and interference avoidance in tdma wireless sensor networks. *Computer Networks*, 53(5) :569–582.

[Mahmood and Jantti, 2009] Mahmood, A. and Jantti, R. (2009). Time synchronization accuracy in real-time wireless sensor networks. IEEE MICC, pages 652–657, Kuala Lumpur, Malaysia.

[Mazzer and Tourancheau, 2009] Mazzer, Y. and Tourancheau, B. (2009). Comparisons of 6lowpan implementations on wireless sensor networks. IARIA SENSORCOMM, pages 689–692, Athens, Greece.

[Merlin and Farber, 1976] Merlin, P. and Farber, D. (1976). Recoverability of communication protocols–implications of a theoretical study. *Communications, IEEE Transactions on*, 24(9) :1036–1043.

[Meseguer, 1992] Meseguer, J. (1992). Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, 96(1) :73–155.

[Milner, 1989] Milner, R. (1989). Communication and concurrency. In *Series in Computer Science. Prentice-Hall International*.

[Milner, 1993] Milner, R. (1993). *The polyadic π-calculus : a tutorial*. Springer.

[Mounier et al., 2007] Mounier, L., Samper, L., and Znaidi, W. (2007). Worst-case lifetime computation of a wireless sensor network by model-checking. PE-WASUN, pages 1–8, Crete Island, Greece.

[Norman et al., 2012] Norman, G., Parker, D., and Sproston, J. (2012). Model checking for probabilistic timed automata. *Formal Methods in System Design*, pages 1–27.

[Olderog and Dierks, 2008] Olderog, E.-R. and Dierks, H. (2008). *Real-time systems : formal specification and automatic verification*. Cambridge University Press.

[Olveczky and Thorvaldsen, 2006] Olveczky, P. and Thorvaldsen, S. (2006). Formal modeling and analysis of wireless sensor network algorithms in Real-Time Maude. IEEE IPDPS, Rhodes Island, Greece.

[Papadimitriou, 2003] Papadimitriou, C. H. (2003). *Computational complexity*. John Wiley and Sons Ltd.

[Pavkovic et al., 2011] Pavkovic, B., Theoleyre, F., and Duda, A. (2011). Multipath opportunistic rpl routing over ieee 802.15.4. ACM MSWiM, pages 179–186, Miami, USA.

[Perkins and Royer, 1999] Perkins, C. E. and Royer, E. M. (1999). Ad-hoc on-demand distance vector routing. IEEE WMCSA, pages 90–100, New Orleans, USA.

[Polastre et al., 2004] Polastre, J., Hill, J., and Culler, D. (2004). Versatile low power media access for wireless sensor networks. ACM SenSys, pages 95–107, Baltimore, USA.

[Proakis, 2001] Proakis, J. G. (2001). Digital communications (ise).

[Proakis and Salchi, 2008] Proakis, J. G. and Salchi, M. (2008). *Digital Communications, 5th ed.* McGrow-Hill.

[Rajendran et al., 2006] Rajendran, V., Obraczka, K., and Garcia-Luna-Aceves, J. J. (2006). Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1) :63–78.

[Ramesh, 2009] Ramesh, M. V. (2009). Real-time wireless sensor network for landslide detection. IARIA SENSORCOMM, pages 405–409, Athens, Greece.

[Rao et al., 2003] Rao, A., Ratnasamy, S., Papadimitriou, C., Shenker, S., and Stoica, I. (2003). Geographic routing without location information. ACM MobiCom, pages 96–108, San Diego, USA.

[Roedig et al., 2006] Roedig, U., Barroso, A., and Sreenan, C. J. (2006). f-mac : A deterministic media access control protocol without time synchronization. EWSN, pages 276–291, Zurich, Switzerland.

[Romdhani, 2012] Romdhani, B. (2012). *Exploitation de l'hétérogénéité des réseaux de capteurs et d'actionneurs dans la conception des protocoles d'auto-organisation et de routage*. These, INSA de Lyon.

[Rowe et al., 2006] Rowe, A., Mangharam, R., and Rajkumar, R. (2006). Rt-link : A time-synchronized link protocol for energy-constrained multi-hop wireless networks. IEEE SECON, pages 402–411, Reston, USA.

[Rumbaugh et al., 1999] Rumbaugh, J., Jacobson, I., and Booch, G. (1999). The unified modeling language reference manual.

[Santinelli and Cucu-Grosjean, 2011] Santinelli, L. and Cucu-Grosjean, L. (2011). Toward probabilistic real-time calculus. *ACM SIGBED Review*, 8(1) :54–61.

[Sardouk et al., 2010] Sardouk, A., Merghem-Boulahia, L., Rahim-Amoud, R., and Gaiti, D. (2010). Data Aggregation in WSNs : a Survey. *Materials Science Research Journal*, 4(4) :327–370.

[Schmitt and Roedig, 2005] Schmitt, J. B. and Roedig, U. (2005). Sensor network calculus–a framework for worst case analysis. IEEE DCOSS, pages 141–154. Los Angeles, USA.

[Schmitt et al., 2007] Schmitt, J. B., Zdarsky, F. a., and Thiele, L. (2007). A Comprehensive Worst-Case Calculus for Wireless Sensor Networks with In-Network Processing. IEEE RTSS, Tucson, USA.

[Seah et al., 2009] Seah, W. K., Eu, Z. A., and Tan, H.-P. (2009). Wireless sensor networks powered by ambient energy harvesting (wsn-heap)-survey and challenges. In *IEEE VITAE*, pages 1–5. IEEE.

[Servetto and Barrenechea, 2002] Servetto, S. D. and Barrenechea, G. (2002). Constrained random walks on random graphs : Routing algorithms for large scale wireless sensor networks. WSNA, pages 12–21, Atlanta, USA.

[Shaikh et al., 2007] Shaikh, F. K., Khelil, A., and Suri, N. (2007). On modeling the reliability of data transport in wireless sensor networks. IEEE PDP, pages 395–402, Naples, Italy.

[Singh et al., 2010] Singh, A., Ramakrishnan, C., and Smolka, S. A. (2010). A process calculus for mobile ad hoc networks. *Science of Computer Programming*, 75(6) :440 – 469.

[Slind and Norrish, 2008] Slind, K. and Norrish, M. (2008). A brief overview of hol4. In *Theorem Proving in Higher Order Logics*, pages 28–32. Springer.

[Sobrinho and a.S. Krishnakumar, 1999] Sobrinho, J. and a.S. Krishnakumar (1999). Quality-of-service in ad hoc carrier sense multiple access wireless networks. *IEEE Journal on Selected Areas in Communications*, 17(8) :1353–1368.

[Sohraby et al., 2007] Sohraby, K., Minoli, D., and Znati, T. (2007). *Wireless sensor networks : technology, protocols, and applications*. John Wiley & Sons.

[Son et al., 2006] Son, D., Krishnamachari, B., and Heidemann, J. (2006). Experimental study of concurrent transmission in wireless sensor networks. ACM SenSys, pages 237–250, Boulder, USA.

[Stankovic and Abdelzaher, 2003] Stankovic, J. and Abdelzaher, T. (2003). Speed : a stateless protocol for real-time communication in sensor networks. IEEE ICDCS, pages 46–55, Providence, USA.

[Sun et al., 2009] Sun, J., Liu, Y., and Dong, J. (2009). Model checking csp revisited : Introducing a process analysis toolkit. ISoLA, pages 307–322, Potsdam, Germany.

[Tan et al., 2010] Tan, R., Xing, G., Chen, J., Song, W.-Z., and Huang, R. (2010). Quality-driven volcanic earthquake detection using wireless sensor networks. IEEE RTSS, pages 271–280, San Diego, CA, USA.

[TI, 2007] TI (2007). 2.4 ghz ieee 802.15.4 / zigbee-ready rf transceiver.

[Tilak et al., 2002] Tilak, S., Abu-Ghazaleh, N. B., and Heinzelman, W. (2002). A taxonomy of wireless micro-sensor network models. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(2) :28–36.

[Tschirner et al., 2008] Tschirner, S., Xuedong, L., and Yi, W. (2008). Model-based validation of qos properties of biomedical sensor networks. ACM EMSOFT, pages 69–78, Atlanta, USA.

[van Dam and Langendoen, 2003] van Dam, T. and Langendoen, K. (2003). An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks. ACM SenSys, pages 171–181, Los Angeles, USA.

[van Hoesel and Havinga, 2004] van Hoesel, L. F. and Havinga, P. (2004). A lightweight medium access protocol (lmac) for wireless sensor networks : Reducing preamble transmissions and transceiver state switches.

[Varga et al., 2001] Varga, A. et al. (2001). The omnet++ discrete event simulation system. ESM, Prague, Czech Republic.

[Wandeler, 2006] Wandeler, E. (2006). *Modular performance analysis and interface-based design for embedded real-time systems*. PhD thesis, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH.

[Wang, 1990] Wang, Y. (1990). Real-time behaviour of asynchronous agents. *CONCUR '90 Theories of Concurrency : Unification and Extension*, 458 :502–520.

[Watteyne, 2008] Watteyne, T. (2008). Energy-efficient slef-organization for wireless sensor networks. Phd thesis manuscript, insa de lyon.

[Watteyne et al., 2007] Watteyne, T., Augé-Blum, I., Dohler, M., and Barthel, D. (2007). Geographic forwarding in wireless sensor networks with loose position-awareness. IEEE PIMRC, pages 1–5, Athens, Greece.

[Watteyne et al., 2009a] Watteyne, T., Augé-Blum, I., Dohler, M., Ubéda, S., and Barthel, D. (2009a). Centroid virtual coordinates - a novel near-shortest path routing paradigm. *Comput. Netw.*, 53.

[Watteyne et al., 2006a] Watteyne, T., Augé-Blum, I., and Ubéda, S. (2006a). Dual-mode real-time mac protocol for wireless sensor networks : a validation/simulation approach. InterSense, page 2, Nice, France.

[Watteyne et al., 2006b] Watteyne, T., Bachir, A., Dohler, M., Barthe, D., and Auge-Blum, I. (2006b). 1-hopmac : An energy-efficient mac protocol for avoiding 1-hop neighborhood knowledge. IEEE SECON, pages 639–644, San Diego, USA.

[Watteyne et al., 2009b] Watteyne, T., Pister, K., Barthel, D., Dohler, M., and Auge-Blum, I. (2009b). Implementation of gradient routing in wireless sensor networks. IEEE GLOBECOM, pages 1–6, Hawaii, USA.

[Wibling et al., 2004] Wibling, O., Parrow, J., and Pears, A. (2004). Automatized Verification of Ad Hoc Routing Protocols. pages 343–358.

[Wikipedia, 2013a] Wikipedia (2013a). http ://en.wikipedia.org/wiki/extreme_ value_theory, viewed 12 septembre 2013.

[Wikipedia, 2013b] Wikipedia (2013b). http ://fr.wikipedia.org/wiki/criticite, viewed 20 septembre 2013.

[Wikipedia, 2013c] Wikipedia (2013c). http ://fr.wikipedia.org/wiki/feu_de_foret, viewed 7 september 2013.

[Wood et al., 2008] Wood, A., Stankovic, J., Virone, G., Selavo, L., He, Z., Cao, Q., Doan, T., Wu, Y., Fang, L., and Stoleru, R. (2008). Context-aware wireless sensor networks for assisted living and residential monitoring. *Network, IEEE*, 22(4) :26–33.

[WSNet, 2009] WSNet (2009). http ://wsnet.gforge.inria.fr/, viewed 6 march 2013.

[Wu et al., 2012] Wu, W., Cao, J., Wu, H., and Li, J. (2012). Robust and dynamic data aggregation in wireless sensor networks : A cross-layer approach. IEEE UIC/ATC, pages 306–313, Fukuoka, Japan.

[Xu et al., 2009] Xu, T., Yuan, L., and Niu, B. (2009). Data fusion algorithm based on event-driven and minimum delay aggregation path in wireless sensor network. In *Emerging Intelligent Computing Technology and Applications. With Aspects of Artificial Intelligence*, pages 1028–1038. Springer.

[Yang, 2010] Yang, F. (2010). Reliable and Time-Constrained Communications in Wireless Sensor Networks. Phd thesis manuscript, insa de lyon.

[Ye et al., 2005a] Ye, F., Zhong, G., Lu, S., and Zhang, L. (2005a). Gradient broadcast : a robust data delivery protocol for large scale sensor networks. *Wirel. Netw.*, 11 :285–298.

[Ye et al., 2005b] Ye, M., Li, C., Chen, G., and Wu, J. (2005b). Eecs : an energy efficient clustering scheme in wireless sensor networks. IEEE IPCCC, pages 535–540, Poenix, USA.

[Ye et al., 2004] Ye, W., Heidemann, J., and Estrin, D. (2004). Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Trans. Netw.*, 12 :493–506.

[Yousefi et al., 2012] Yousefi, H., Yeganeh, M. H., Alinaghipour, N., and Movaghar, A. (2012). Structure-free real-time data aggregation in wireless sensor networks. *Computer Communications*, 35(9) :1132–1140.

[Zhang et al., 2007] Zhang, J., Li, W., Han, N., and Kan, J. (2007). Forest fire detection system based on a zigbee wireless sensor network. In *Journal of Beijing Forestry University, vol. 29, no. 4*, pages 369–374.

[Zhang and Gorce, 2007] Zhang, R. and Gorce, J.-M. (2007). Connectivity of wireless sensor networks with unreliable links. IEEE CHINACOM, pages 866–870, Shanghai, China.

[Zhao et al., 2007] Zhao, Y., Chen, Y., Li, B., and Zhang, Q. (2007). Hop id : a virtual coordinate based routing for sparse mobile ad hoc networks. *Mobile Computing, IEEE Transactions on*, 6(9) :1075–1089.

[Zuniga and Krishnamachari, 2004] Zuniga, M. and Krishnamachari, B. (2004). Analyzing the transitional region in low power wireless links. IEEE SECON, pages 517–526, Santa Clara, USA.