INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche LILLE - NORD EUROPE

Université Lille1 Sciences et Technologies

cnrs dépasser les frontières

UNIVERSITÉ LILLE 1

Numéro d'ordre : 40190

# Contrôle de topologie dans les réseaux de capteurs : de la théorie à la pratique

*Thèse de doctorat (spécialité Informatique) présentée le 15 décembre 2009*

*par* FADILA KHADAR

**Composition du jury**

| | |
|---|---|
| *Rapporteurs :* | Andrzej Duda, Professeur, INP-ENSIMAG, Grenoble |
| | Isabelle Guérin Lassous, Professeur, Université Lyon I, Lyon |
| *Examinateurs :* | Serge Fdida, Professeur, Université Pierre et Marie Curie, Paris |
| | Aline Carneiro Viana, Chargée de recherche, INRIA Saclay |
| | Nathalie Mitton, Chargée de recherche, INRIA Lille ? Nord Europe |
| *Directeur :* | David Simplot-Ryl, LIFL, Université des Sciences et Technologies de Lille |

.

*Merci à David et à tant d'autres pour leur patience...*

# Table des matières

# Introduction

# 1

« Radio has no future. »
« Wireless [telegraphy] is all very well but I'd rather send a message by a boy on a pony ! »

**Lord Kelvin (Sir William Thomson)**,
*Physicien britannique à l'origine du zéro absolu.*

---

Sommaire



---

Contrairement à l'affirmation de Lord Kelvin, la radio est très utilisée de nos jours. Les premières utilisations se sont faites dans le cadre d'un émetteur unique à fortes capacités et de plusieurs récepteurs ou terminaux (télévision, radiodiffusion, ...). Par la suite, les communications sont devenues bidirectionnelles, permettant à des technologies telles que la téléphonie et le wifi de prendre leur essor. Le prochain défi est de permettre à tous les éléments du réseau de acheminer de l'information et de pouvoir ensuite construire des réseaux *ad-hoc*, c'est à dire des réseaux dont le déploiement ne nécessite aucune infrastructure. La baisse des coûts des composants informatiques et leur miniaturisation a permis l'émergence de nouveaux types de réseaux : les réseaux de capteurs sans fil. Ils devraient, à terme, être utilisés pour de nombreuses applications telles que la surveillance de zone, la détection d'intrusion, etc. Les réseaux de capteurs sont des réseaux *ad-hoc* particulier qui présentent des objectifs et des contraintes différentes. En effet, leur capacité en terme de calcul, mémoire et batterie est très faible et le paradigme de communication qu'il présente est différent de celui des réseaux *ad-hoc* classiques. Dans ce chapitre, nous introduisons la technologie des réseaux de capteurs et présentons les différents défis qu'ils amènent. Dans cette optique, nous décrivons dans un premier temps le matériel utilisé. Nous passons ensuite en revue les différentes applications envisagées avant de détailler les problématiques associées à ce type de réseaux. Enfin, nous présentons le rôle de la pile de communication dans ces réseaux avant de discuter des objectifs de nos travaux.

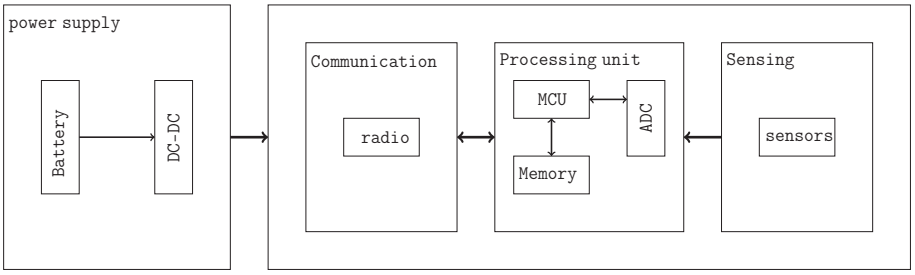



FIGURE 1.1 – *Structure générique d'un capteur*

| **nom** | 802.15.4 | **bande de fréquence** | **sensibilité** |
|---|---|---|---|
| TI CC2520 | oui | 2.4GHz | −98dBm |
| TI CC2420 | oui | 2.4GHz | −95dBm |
| TI CC1101 | non | 868/915MHz | −94dBm |
| TI CC2500 | non | 2.4GHz | −89dBm |
| Atmel AT86RF230 | oui | 2.4GHz | −89dBm |

FIGURE 1.2 – *Exemple de chip radios avec bande de fréquence et sensibilité respectives.*

## 1.1 Les réseaux de capteurs

Un capteur est une entité à capacités de calcul et de mémoire limitées capable d'obtenir des informations sur son environnement. La température, la pression, la luminosité, ou la présence d'un gaz sont des exemples d'informations qu'un capteur est capable de recueillir. Il est muni d'un système de communication sans fil lui permettant de transmettre les données recueillies. Prévu pour être déployé dans des environnements hostiles ou difficiles d'accès, il embarque une batterie dont le remplacement est rarement possible. Un capteur doit donc utilisé celle-ci avec parcimonie de manière à prolonger sa durée de vie. La figure 1.1 reprend l'architecture générale d'un capteur. Il se compose d'un unité de communication, d'une unité de traitement, et d'une unité d'alimentation. L'unité de traitement constitue le cœur du capteur. Elle se compose d'un micro-contrôleur, d'une mémoire (souvent très réduite) et d'un ADC (*Analog To Digital Converter*) permettant de transformer les données analogiques recueillies en signaux numériques. L'unité de communication permet au capteur d'envoyer ses données à l'extérieur sans liaison filaire. Pour ce faire, plusieurs solutions sont envisageables bien que celle la plus fréquemment retenue soit la radio. On peut citer l'exemple des capteurs *Smart Dust* [46] qui utilise des media optiques. Ceux-ci sont moins coûteux en énergie mais pour que deux capteurs puissent communiquer, il faut qu'il n'y ait aucun obstacle entre eux. Le tableau 1.2 reprend des exemples de chips radios ainsi que certaines de leur caractéristiques.

Un réseau de capteurs est un ensemble de capteurs qui coopèrent ensemble afin d'accomplir une tâche commune. Cette tâche peut par exemple être la surveillance d'une zone à risque ou difficile d'accès. Dans ce cas, chaque capteur accepte de relayer les informations recueillies par d'autres capteurs afin de les acheminer jusqu'au puits, c'est-à-dire l'endroit où elles vont être traitées. Pour être viable, la plupart des applications des réseaux de capteurs comptent sur un nombre important de capteurs dont le coût unitaire serait faible. Or, un faible coût implique des capacités de calcul et de mémoire limitées. Elles imposent également l'utilisation de radio de puissance raisonnable, incapable de communiquer l'information directement où elle doit être traitée. L'un des points critiques matériels est également la batterie. Si la zone surveillée est

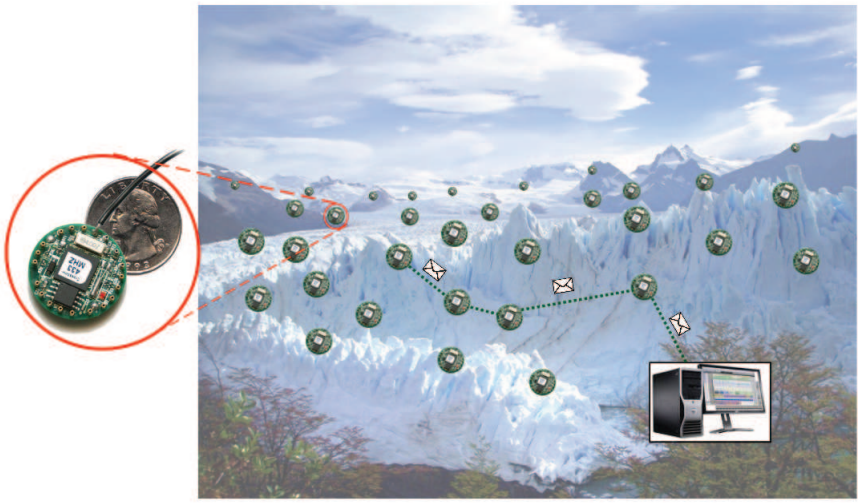


FIGURE 1.3 – *Exemple de réseau de capteurs.*

accidentée ou difficile d'accès, il n'est pas toujours faisable de remplacer les batteries. Le défi le plus important dans les réseaux de capteurs est donc de conserver l'énergie. La radio étant un des composants le plus gourmand en terme d'énergie dans un capteur, son utilisation doit être minimisée, tout en garantissant que les messages soient toujours acheminés à leur destinataire. Lorsque nous parlons d'un capteur appartenant à un réseau, on emploie indifféremment les termes de capteurs et de *nœuds*.
Pour gérer les communications entre capteurs, l'approche généralement retenues est la même que dans les réseaux classiques, à savoir la constitution d'une pile de communication dans laquelle chaque couche possède un rôle particulier. Dans la section suivante, nous expliquons le rôle de cette pile de communication.

## 1.2 Rôle de la pile de communication

Une pile de communication type pour capteur est montrée sur la figure 1.4. La première étape consiste à permettre à des capteurs adjacents de communiquer en utilisant la radio. C'est le rôle de la **couche physique**. Elle sélectionne la fréquence d'émission. Elle est également responsable de la détection du signal, de la modulation, de la conversion des bits en signaux électriques ou optiques, et inversement.
Une fois qu'il est possible pour des paires de capteurs suffisamment proches de communiquer, il faut s'assurer que ceux qui se trouvent dans la même zone n'essaient pas d'émettre en même temps. En effet, cela pourrait causer des collisions. Il faut donc contrôler l'accès au médium pour des nœuds relativement proches.

Le **protocole MAC** (Medium Access Protocol) est en charge de la régulation de cet accès au médium pour éviter au maximum les collisions. Il doit donc s'assurer la fiabilité des communications point-à-point, maximiser l'utilisation de la bande passante, assurer l'équité dans l'attribution de la bande passante, minimiser la latence, et en priorité minimiser la consommation d'énergie. Cependant, il n'est pas responsable de contrôler les données destinées aux couches supérieures.
Il s'agit du rôle du **protocole de liaison de données** qui assure le découpage des données en paquets. La taille des paquets a une influence non négligeable sur les performances du réseau. Elle assure également le contrôle d'erreur qui consiste à détecter les paquets non reçus par le destinataire ou trop erronés pour être utilisable. Elle peut aussi être responsable de la gestion de liens : découverte, mise en place et de la maintenance des liens entre nœuds voisins.
Cet ensemble de protocoles permet donc de gérer les communications point-à-point. Si deux

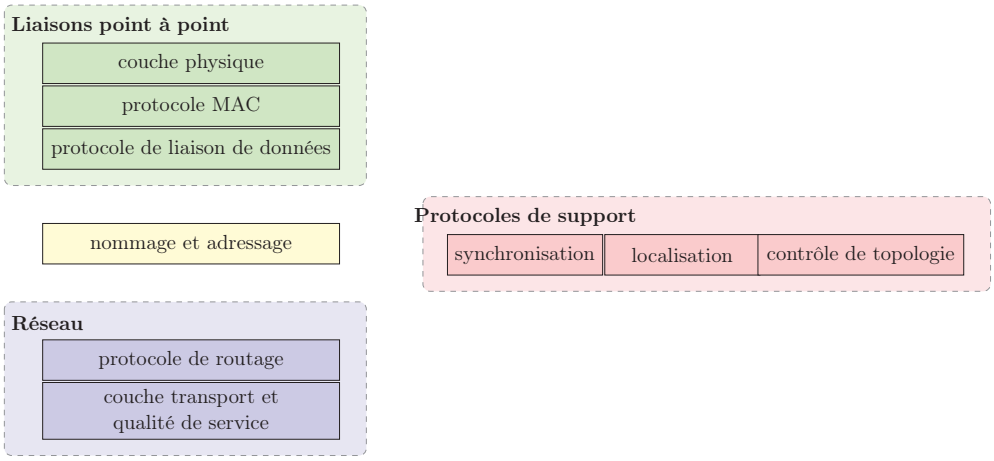



FIGURE 1.4 – *Pile de communication d'un capteur. Elle comprend, pour la gestion des communications point à point, une couche physique, un protocole MAC et un protocole de liaisons de données. Une entité est responsable du nommage et de l'adressage. La gestion des communications de bout en bout est opérée par le protocole de routage et une couche transport. Cette pile peut éventuellement comprendre des protocole de support fournissant la synchronisation, la localisation et le positionnement ou le contrôle de topologie.*

nœuds souhaitant communiquer ne sont pas à portée de communication directe, il faut être capable de sélectionner un certain nombre de relais responsable d'acheminer le message jusqu'à sa destination. Le **protocole de routage** permet à un nœud de déterminer le chemin à emprunter pour envoyer un message à un nœud donné (le puits par exemple). Son rôle est donc essentiel : il a un effet important sur la fiabilité du réseau, la faible consommation énergétique des nœuds et plus généralement sur l'efficacité du réseau. L'utilisation d'un protocole de routage suppose donc qu'il existe un moyen d'identifier chaque nœud. L'entité de **nommage et d'adressage** est responsable d'attribuer à chaque nœud un identifiant permettant de faciliter les opérations de routage. Comme le protocole MAC, le protocole de routage ne s'occupe que d'acheminer les données au destinataire, sans en vérifier la consistance.
La **couche transport** gère les communications de bout en bout (contrôle d'erreurs, contrôle de flux,découpage en fragments pour les messages volumineux,etc). Elle assure ainsi la bonne reception des paquets et la qualité de service.
Pour être capable d'assurer leur fonctions, ces protocoles peuvent requérir des informations particulières, telle que la position des nœuds pour un routage utilisant la position géographique pour sélectionner le prochain relais. Ces informations sont fournies par les **protocoles de support**. Par exemple, il peut exister un protocole de support s'assurant de la synchronisation entre les nœuds du réseau, c'est-à-dire vérifiant que chaque nœud possède le même temps que les autres nœuds du réseau.
Si le capteur n'est pas équipé d'un GPS (coûteux autant au point de vue pécunier qu'un niveau énergétique), un protocole peut lui fournir une approximation de sa position, ou de la position d'autres nœuds du réseau.
Lors du déploiement d'un réseau de capteurs, il peut être nécessaire de déployer un nombre important de capteurs pour être certains de recouvrir une zone donnée. Cependant, dans cet éventualité, chaque nœud possède un nombre conséquent de voisins dans sa zone de communication. Dans la plupart des piles de communication, chaque nœud prend connaissance de ce voisinage et maintient des informations sur chacun de ces voisins, ce qui est coûteux en énergie et en espace mémoire.
Le **contrôle de topologie** consiste donc à gérer les relations de voisinage et ainsi éliminer les liens considérés comme redondants. Cette élimination peut s'effectuer de diverses manières comme

par l'établissement de hiérarchies : un nœud est responsable d'un ensemble de capteurs et en contrôle les communications avec les nœuds d'autres zones. Une autre manière est de limiter la portée de communication de chaque nœud, c'est à dire de réduire sa zone de communication.

## 1.3 Objectifs de la thèse

Dans ce document, nous présentons nos travaux concernant le contrôle de topologie par réduction de graphe : le réseau de capteurs est représenté par un graphe dont les sommets représentent les nœuds du réseau et les arêtes dénote une possibilité de communication entre deux nœuds. La réduction de graphe consiste donc à appliquer un algorithme de réduction de graphe (qui enlève les liens redondants) et ne conserver que les voisins dans ce graphe réduit. Cette algorithme de réduction a pour seule contrainte de maintenir la connexité du graphe. Dans la littérature, de nombreux travaux sur les réductions de graphe existent. Cependant, ils utilisent principalement un modèle selon lequel deux nœuds reliés par une arête dans le graphe peuvent toujours communiquer,*i.e.,* jamais une communication n'échouera entre ces deux nœuds. Nos travaux ont donc pour objectif de réutiliser ces travaux et de les adapter pour une utilisation pratique. La démarche que nous adoptons pour cela est la suivante. Dans un premier temps, nous adaptons ces algorithmes en utilisant un modèle de couche physique plus réaliste : si un lien existe entre deux nœuds, cela signifie qu'une communication entre ces deux nœuds a une certaine probabilité de réussir. Nous étudions dans les mêmes conditions les algorithmes d'assignement de portée qui consiste à choisir pour chaque nœud l'importance de sa zone de communication. Nous montrons ensuite comment adapter ces concepts à une implémentation réelle en proposant une pile de communication dans laquelle est ensuite intégré le contrôle de topologie.

## 1.4 Organisation du document

Ce document présente notre étude du contrôle de topologie. Le chapitre 2 est consacré à une revue de l'état de l'art concernant les piles de communication et système d'exploitation pour réseau de capteurs. Nous y présentons également les différents modèles de couche physique utilisés ainsi qu'une revue des différentes solutions existantes pour effectuer du contrôle de topologie.
Les chapitres 3, 4 et 5 présentent les contributions apportées. Le chapitre 3 présente les différents algorithmes de contrôle de topologie dans le cadre de l'utilisation d'une couche physique réaliste. Le chapitre 4 décrit Goliath, la pile de communication que nous avons implémentée et les concepts qui y sont mis en œuvre. Dans le chapitre 5 nous montrons comment nous avons inclut le contrôle de topologie dans Goliath et montrons les bénéfices apportés par cette solution. Le chapitre 6 nous permettra de conclure ce travail et de présenter d'éventuelles pistes de recherche faisant suite à ces travaux.

# État de l'Art

# 2

Sommaire



Dans ce chapitre, nous présentons dans un premier temps différentes piles de communication et systèmes d'exploitation pour capteurs. Nous décrivons ensuite les hypothèses et modèles utilisés dans ce document. Le contrôle de topologie vise à réduire l'ensemble des voisins logiques d'un nœud, tandis que le contrôle de portée vise à sélectionner la meilleure portée pour atteindre ces voisins tout en réduisant l'ensemble physique des voisins d'un nœud. Le contrôle de puissance de transmission tel que nous le définissons vise à mettre en concordance le voisinage physique et le voisinage logique d'un nœud. Nous décrivons dans ce chapitre les différents approches existantes concernant le contrôle de topologie et le contrôle de puissance.

## 2.1 Systèmes d'exploitation et pile de communication pour capteurs

Plusieurs systèmes d'exploitation pour capteurs (ou plus généralement pour systèmes embarqués) ont été implémentés, chacun offrant des fonctionnalités propres. Dans cette section nous présentons différents systèmes d'exploitation couramment utilisés lors de la programmation de capteurs. Le système que nous avons choisi d'utiliser pour l'implémentation de GoliatHest FreeRTOS. Outre une brève description dans cette section, nous fournissons de plus amples détails sur ce système dans le chapitre 4. Le rôle d'un système d'exploitation pour capteurs est plus restreint que les systèmes d'exploitation dits classiques. Il fournit généralement une abstraction matérielle et éventuellement un ordonnanceur. Nous décrivons également les piles de communication associées à ces systèmes d'exploitation, ainsi que le standard ZigBee proposé par la ZigBee Alliance. D'autres piles existent telles que nanoStack 6lowPAN existent mais nous choisissons de ne pas les décrire étant donné que leur particularité (intégrer IPv6) n'est pas abordé dans ce document.

### 2.1.1 TinyOS

TinyOS [27] est un système d'exploitation gratuit et open-source dédié aux capteurs dont la sortie date de 2000. Il supporte actuellement les plateformes suivantes : eyesIFXv2, intelmote2, mica2, mica2dot, micaZ, telosb, tinynode et btnode3. Les applications pour TinyOS sont écrites en nesC [23] (Network Embedded System C), une extension du C. Il adopte une programmation par évènement, c'est-à-dire que le système attend que des évènements se produisent. Ce paradigme de programmation permet de ne pas avoir une pile d'exécution par processus, et donc de ne pas avoir à sauvegarder leur contexte d'exécution. Deux types d'entités sont ordonnançables : les évènements et les tâches. Un évènement peut être l'arrivée d'un paquet, la fin d'un timer ou une nouvelle donnée disponible. La programmation se fait par composants, chaque composant fournit des commandes et réagit à des évènements. Chaque appel à une commande ou réaction à un évènement s'éxecute jusqu'à son achèvement. Il n'y a aucune préemption ce qui permet d'éviter les conflits entre tâches. La communication entre composants se fait par l'utilisation d'interfaces : un composant requiert certaines interfaces et en fournit d'autres. L'ordonnancement dans TinyOS se fait de manière non préemptive et adopte une stratégie de FIFO : le prochain processus sera celui arrivé le premier.Il existe plusieurs extensions de TinyOS comme TinyDB (Tiny DataBase, permet d'interroger le réseau comme une base de données), Maté (machine virtuelle au-dessus de TinyOS permettant le chargement dynamique de code), etc.
La couche réseau fournit une interface appelée *Active Messages* (AM). Il s'agit de petits paquets auxquels sont associés un octet identifiant. Lors de la réception d'un tel message, un événement est déclenché au niveau des handlers enregistré pour la réception de messages ce type. AM fournit de base un protocole d'envoi non fiable de messages à un sauts. De nombreuses piles protocolaires ont été implémentées sur TinyOS [4].

### 2.1.2 Contiki et Rime

Contiki [15] est également open-source et multitâche et utilise aussi la programmation événementielle. En plus du noyau, un système utilisant Contiki contient des processus, qui peuvent être des applications ou des services, c'est-à-dire un processus proposant des fonctionnalités à une ou plusieurs applications. La communication entre processus se fait par l'envoi d'événements. Les appels aux gestionnaires d'événements se font également jusqu'à leur achèvement, l'ordonnanceur ne fournissant pas de possibilité de préemption pour ceux-ci. Il est possible d'obtenir un environnement multi-threadé et préemptif. Contiki fournit également la possibilité de charger dynamiquement des processus, et permet également leur remplacement après déploiement. Cette

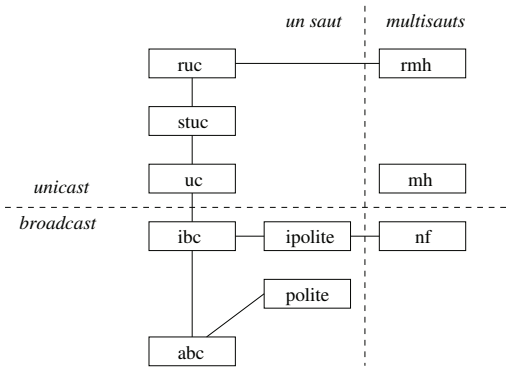



FIGURE 2.1 – *Pile de communication* Rime.

fonctionnalité est fournie par une bibliothèque au dessus du noyau. Ce système d'exploitation utilise le concept de *protothread* [17] qui permettent d'écrire des programmes à évènement comme des threads et ce avec un faible surcoût (2 octets et quelques cycles de processeur). L'empreinte mémoire de Contiki est plus importante que celle de TinyOS ([15]), mais il rend possible l'ordonnancement avec priorité et fournit une plus grande flexibilité comme le chargement dynamique d'applications.
Contiki fournit deux piles de communication. La première, uIP, permet d'utiliser TCP/IP sur des architectures 8 bits [14]. Rime [16] est la seconde pile de communication proposée par Contiki. Elle offre un ensemble de primitives de communication (représentée sur la figure 2.1) utilisables par les applications et les protocoles au-dessus de la pile protocolaire. La primitive la plus basique est le broadcast anonyme à un saut (appelée *abc*). Elle est utilisée par toutes les autres primitives. Le paquet est envoyé à tous les voisins physiques du nœud qui sont en train d'écouter le medium, sans contenir d'information sur l'émetteur. La primitive *ibc* effectue la même tâche mais en incluant dans le paquet l'adresse de l'émetteur. L'envoi de paquet en unicast se fait à l'aide de la primitive *uc*, qui s'appuie sur *ibc* et y ajoute l'adresse du destinataire. Ce module sert également à écarter les paquets dont le nœud n'est pas le destinataire. Elle est à la base de la primitive *suc* (Stubborn single-hop unicast) qui envoie un paquet continuellement jusqu'à ce qu'une couche supérieure annule la transmission. La primitive *suc* permet à la primitive *ruc* (Reliable Single-hop Unicast) de proposer un unicast à un saut fiable. Elle décide du nombre de réémission d'un paquet. La primitive *polite* envoie en broadcast à un saut un paquet dans un intervalle de temps donné si aucun paquet avec la même en-tête n'a été reçu dans cet intervalle. Il est utilisé pour implémenter des algorithmes de gossip. *ipolite* est une variante de *polite* dans laquelle l'information sur l'émetteur du paquet est ajoutée. Toutes ces primitives permettent des communications à un saut. La primitive *mh* et sa variante fiable *rmh* fournissent les outils pour effectuer du multisauts. Ceux ne sont cependant pas des algorithmes de routage. A chaque fois qu'un paquet est reçu, ces primitives utilisent la fonction fournie par les modules qui de plus haut niveau pour déterminer le prochain saut. Le flooding est fourni par le module *nf*.
Pour déterminer le format des paquets, Rime utilise Chameleon dont le but est de séparer la logique de communication et les en-têtes de paquets. L'idée est de dissocier la construction et la lecture des en-têtes de la pile de communication de leur signification. Les protocoles utilisant Chameleon utilisent ou définissent des *attributs de paquets* en y indiquant la taille ou le type et leur portée, c'est-à-dire s'ils doivent être conservés uniquement au niveau du nœud, envoyés dans les paquets mais conservés seulement à un saut, ou envoyés et conservés jusqu'à leur destination en multisauts.

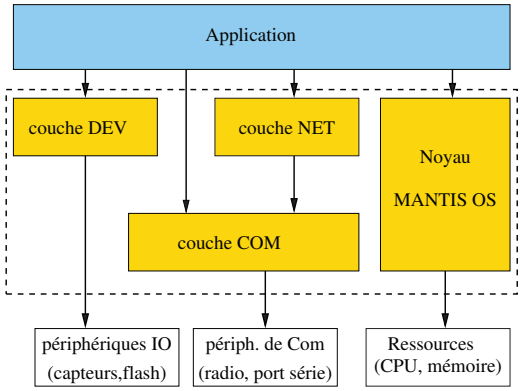



FIGURE 2.2 – *Vue générale du système d'exploitation MANTIS OS.*

### 2.1.3 MANTIS OS

MANTIS (MultimodAl NeTworks of In-situ micro Sensor) OS est un système d'exploitation léger et multi-threadé pour capteurs. Son architecture est montrée figure 2.2. Il supporte plusieurs plateformes parmi lesquelles MICA2, MICAz et les motes TELOS. La programmation d'application sur MANTIS OS se fait en C. Le noyau offre un sous-ensemble de l'API POSIX des threads. Outre le noyau, les trois principaux composants de MANTIS OS sont la couche DEV, la couche NET et la couche COM. La couche DEV fournit un accès aux périphériques d'entrées-sorties. C'est grâce à cette API que le programmeur accède à la flash, au capteurs environnementaux, etc. La couche COM permet d'accèder aux périphériques de communication tels que le port série et la radio. C'est à ce niveau que se trouve la couche MAC et que sont gérés le buffering des paquets et les fonctions de synchronisation.
La pile de communication est gérée au niveau utilisateur dans la couche NET. Par défaut, un module de flooding est fourni comme protocole de routage.

### 2.1.4 FreeRTOS

FreeRTOS est un noyau temps réel pour systèmes embarqués créé par Richard Barry [20]. Il est hautement configurable, permettant d'activer la préemption ou d'utiliser un mode coopératif. L'ordonnancement se fait à intervalles réguliers ou lors d'événements asynchrones. Les tâches de plus haute sécurité s'éxecutent en premier, et en round robin s'il existe plusieurs tâches de même priorité. Une tâche est un processus muni de son propre contexte d'exécution. Il peut donc exister plusieurs tâches dans une application, chacune possédant une priorité définie à sa création. La communication entre tâches peut s'effectuer l'utilisation de file, de sémaphores ou des mutex pour partager une ressource communes. Une plus ample description de FreeRTOS est disponible au chapitre 4.

### 2.1.5 ZigBee

La pile de communication ZigBee vise les petits appareils à faible coût et faible capacité. Les taux de transfert sont de 250 ko/s, 100 ko/s, 40 ko/s et 20 ko/s. Les bandes de fréquence utilisables sont celles de 2450 MHz (16 canaux), 915 MHz (30 canaux) et 868 MHz (3 canaux). L'architecture de la pile de communication ZigBee est montrée sur la figure 2.3. Elle utilise pour les couches physique et d'accès au medium les couches définies par IEEE 802.15.4. Les couches définies par la ZigBee alliance sont les couches réseau et applicative.

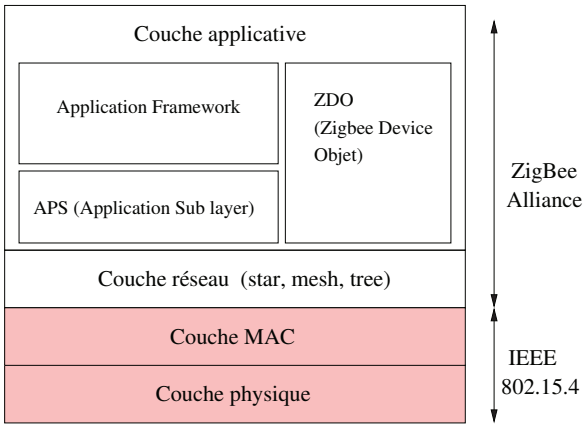



FIGURE 2.3 – *La pile protocolaire définie par la ZigBee Alliance.*

### IEEE 802.15.4

Ce standard [1] définit une couche physique (PHY) et une couche d'accès au medium (MAC). Un nœud du réseau peut implémenter toutes les fonctionnalités de 802.15.4 (on parle alors de *Full-Function Device* ou FFD) ou n'en implémenter qu'une partie (il s'agit d'un *Reduced-Function Device* ou RFD). Un réseau IEEE 802.15.4 peut avoir deux topologies différentes : en étoile ou pair-à-pair. Lorsqu'il s'agit d'une topologie en étoile, un FFD joue le rôle de coordinateur. Toutes les communications passent par ce nœud. La mise en place d'une telle topologie se fait par l'activation d'un nœud FFD qui peut alors construire son propre réseau et devenir coordinateur. Des FFD ou RFD peuvent alors demander au coordinateur à rejoindre le réseau. Dans un réseau pair-à-pair, chaque nœud peut communiquer avec un nœud se trouvant dans sa zone de communication. Le coordinateur est désigné, parce qu'il est le premier à émettre par exemple. Cette topologie permet de construire d'autres topologies plus complexes.
Un réseau 802.15.4 peut fonctionner avec ou sans frames servant de balises (appelées *balises*). Lorsque les beacons ne sont pas utilisés, l'accès au medium se fait par la méthode CSMA/CA non slottée, et le coordinateur reste toujours en écoute d'éventuelles transmissions. L'utilisation de beacons offrent plus de possibilités, puisqu'elle permet l'utilisation de superframe. Une superframe est la période entre deux balises et indique l'activité du coordinateur. Sa structure est donnée au nœuds du réseau dans la balise. Le coordinateur peut alors choisir d'avoir une période d'inactivité dans la superframe et se mettre en veille. Les nœuds accèdent au medium en utilisant la méthode CSMA-CA slottée. Le coordinateur peut également choisir d'allouer des slots sans contention dans la superframe (ils sont alors appelés *Guaranteed Time Slots* ou GTSs). Les nœuds envoyant des données au coordinateur peuvent demander un accusé de réception. Les transferts de données du coordinateur aux nœuds du réseau se font différemment selon que les balises sont activées ou non. Lorsque les balises sont activées, le coordinateur indique dans la balise qu'il y a des données disponibles pour le nœud destinataire. Ce dernier peut alors demander à les recevoir. Lorsque l'envoi de balises est désactivé, le coordinateur conserve les données jusqu'à ce que le nœud destinataire en fasse la demande.

### Le standard ZigBee

En définitive, le standard ZigBee ne spécifie que les couches les plus hautes, c'est-à-dire la couche réseau et la couche applicative. Un nœud du réseau peut avoir un de ces trois rôles : terminal (RFD ou FFD), routeur (FFD avec possibilités de router des paquets) ou coordinateur (FFD contrôlant tout le réseau). La couche réseau permet de construire des topologies plus complexes que celle en

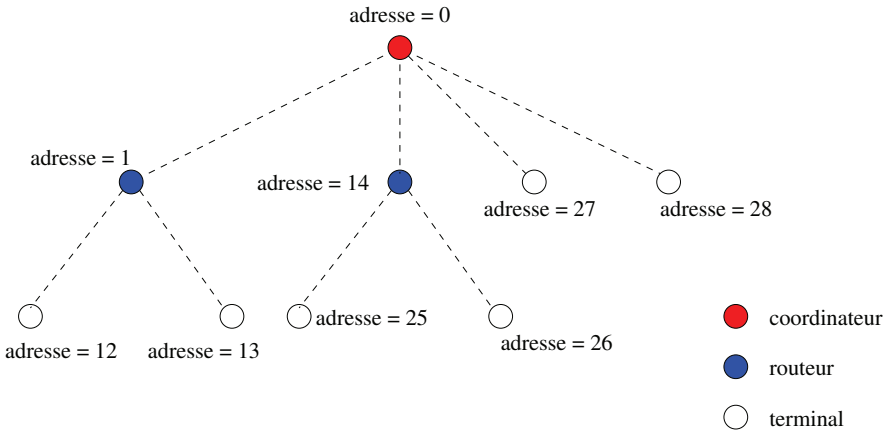



FIGURE 2.4 – *Adressage dans ZigBee.*

étoile (par exemple un arbre ou un réseau mesh). Elle fournit également le routage multisauts, la découverte de routes et leur maintenance, la sécurité et le mécanisme de raccordement à un réseau.
La formation du réseau se fait de manière incrémentale. Lorsqu'un nœud désire rejoindre un réseau, il lance une procédure de découverte de réseau qui s'appuie sur l'envoi régulier de messages par les routeurs du réseau. Une fois le choix fait entre les différents réseaux disponibles, le nœud sélectionne un père parmi les nœuds dont il a connaissance, et demande à la couche MAC d'initier une procédure d'association. Il obtient ainsi un identifiant auprès de son père qui sera utilisé pour toutes les communications dans le réseau. L'assignement d'adresse se fait en assignant à chaque routeur un « espace d'adressage ». L'identifiant est sur 16 bits. Le premier octet correspond à l'adresse du nœud. Le deuxième, pour les nœuds ayant le rôle de routeur, représente la dernière adresse qu'il peut assigner à ses fils. Ainsi, un nœud ayant l'identifiant 1 : 14 a l'adresse 1 et peut donner à ses fils des identifiants allant de 2 à 14. Le coordinateur du réseau calcule la taille des plages d'adresse à assigner en fonction du nombre maximum de routeurs et de terminaux dans le réseau, et de la profondeur maximale de l'arbre. Le routage est quant à lui dépendant de la topologie du réseau. Si la topologie utilisée est l'arbre, le routeur profite de la manière d'assigner les adresses pour savoir s'il doit envoyé le message à son père dans l'arbre ou si le nœud destinataire se situe dans l'arbre dont il est la racine. S'il s'agit d'un réseau mesh, les routeurs maintiennent une table de routage dans laquelle une entrée est constituée de la destination, du prochain saut pour atteindre cette destination et de l'état de la route (active, en cours de découverte, inactive). La découverte de route est basée sur AODV (Ad hoc On Demand Distance Vector routing [36]).
Une application telle que définie par ZigBee est un ensemble d'objets applicatifs (*Application Objects* ou APO) répartis dans le réseau, et qui ont localement un identifiant unique. Un APO peut interagir avec d'autres APO dans le réseau en utilisant cet identifiant comme extension de l'adresse du nœud. Le ZDO (*ZigBee Device Object*) est un APO particulier offrant la découverte d'objets et de services dans le réseau, et la sécurité des communications. Le transfert de données est fourni par la sous-couche applicative (*Application Sublayer* ou APS).
Etant donnée la complexité de ces piles de communication (et donc de l'ajout ou la modification de fonctionnalités), et la difficulté d'isoler les effets qu'auraient une implémentation des solutions proposées pour le contrôle de topologie et l'ajustement de portée, nous avons décidé d'implémenter notre propre pile de communication (GOLIATH). Celle-ci est décrite au chapitre 4. Pour

l'implémentation de cette pile, nous utilisons le système d'exploitation FreeRTOS décrit plus en détails dans la section suivante.

## 2.2 Détails sur FreeRTOS

FreeRTOS est un noyau temps réel pour systèmes embarqués créé par Richard Barry supportant des architectures variables allant du processeur 8 bits au processeur 32 bits. On peut notamment citer l'ARM7, l'ARM9, le MSP430 et l'AVR. Il est hautement configurable, permettant d'activer la préemption ou d'utiliser un mode coopératif.

**Ordonnanceur**

L'ordonnanceur a pour rôle de décider quelle tâche doit se trouver en cours d'exécution à un temps donné. L'ordonnancement se fait à intervalles réguliers (à chaque tique temporel, généralement 1 ms) ou lors d'événements asynchrones (disponibilité d'une ressource par exemple). Les tâches de plus haute priorité s'éxecutent en premier, et en round robin s'il existe plusieurs tâches de même priorité. Lorsque la préemption est autorisée, l'ordonnanceur peut stopper l'exécution d'une tâche pour permettre à une tâche plus haute priorité de traiter une interruption ou pour partager l'utilisation des ressources par plusieurs tâches de même priorité.

### 2.2.1 Tâches dans FreeRTOS

Une tâche est un processus muni de son propre contexte d'exécution. Il peut donc exister plusieurs tâches dans une application, chacune possédant une priorité définie à sa création et un nom. Une tâche est généralement constituée d'une fonction dans laquelle se trouve une boucle infinie. Comme illustré sur la figure 2.5, une tâche peut se trouver dans quatre états différents : *prête*, *en cours d'exécution*, *bloquée*, ou *suspendue*. Lors de sa création, la tâche se trouve dans l'état *prête*. A son démarrage, l'ordonnanceur choisit parmi les tâches *prêtes* laquelle exécuter en premier selon leur priorité. Elle peut être suspendue par l'appel de la fonction de suspension de tâche fournie dans l'API de FreeRTOS. Cet appel peut se faire dans la tâche elle-même ou peut-être fait par une autre tâche. Lorsqu'une tâche est suspendue, elle n'est plus disponible pour l'ordonnancement et retourne dans l'état *prête* par un appel explicite.
Une tâche peut passer dans l'état *bloquée* pour différentes raisons. Elle peut par exemple demandée d'être bloquée pendant un délai donné. A l'expiration de ce délai, elle repassera dans l'état *prête*. Elle peut également être en attente sur un sémaphore, un mutex ou une file. Dans tous ces cas, un évènement est déclenché lorsque la ressource est disponible ou le temps d'attente maximum est atteint. Seul un évènement peut faire passer la tâche dans l'état *prête*. Il peut s'agir d'une interruption, d'un événement déclenché par une autre tâche ou d'un évènement temporel indiquant la fin d'un temps d'attente.
Une seule tâche peut se trouver dans l'état *en cours d'exécution* à un instant particulier. Lorsqu'aucune tâche n'est disponible (*i.e.,* aucune tâche n'est dans l'état *prête*), une tâche, appelée *idle hook*, est appelée. Elle possède la priorité la plus basse possible, est toujours dans l'état *prête* et est non bloquante. Elle permet d'effectuer des actions d'économie d'énergie telles que la mise en veille.

### 2.2.2 Communication entre tâches

La communication entre tâches s'effectue par l'utilisation de file, de sémaphores ou de mutex pour partager une ressource commune.

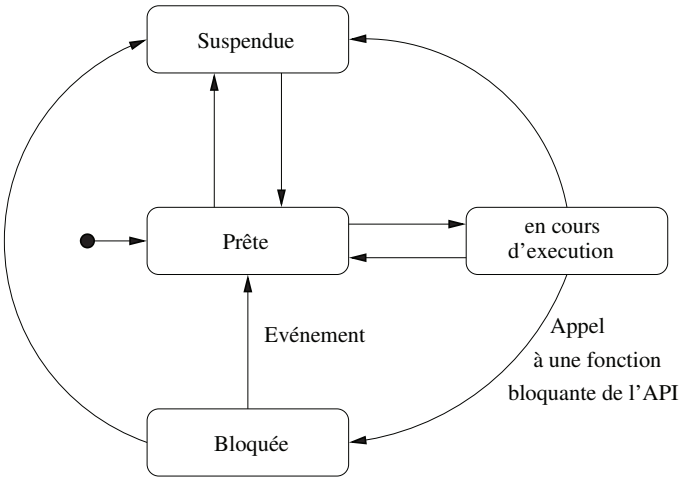



FIGURE 2.5 – *Différents états possibles pour une tâche.*

Lorsque deux tâches souhaitent échanger des données, elles utilisent des files d'attente. Ces dernières peuvent également être utilisées pour la communication entre interruptions et tâches. Le type de données et le nombre d'items qu'elles peuvent contenir sont définis à leur création. Comme les items y sont placés par copie, il est important de minimiser leur taille. Nous nous efforçons donc de n'y mettre que des items d'un octet. La tâche peut spécifier un temps d'attente sur la file. Elle est alors débloquée quoi qu'il arrive à l'expiration de ce délai. Si plusieurs tâches sont en attente sur la même file, la tâche avec la priorité la plus élevée est débloquée en premier. Nous utilisons également les sémaphores binaires pour gérer l'exclusion mutuelle et la synchronisation. Les sémaphores binaires sont des files d'attente avec un seul item. La file est alors vide ou pleine, la valeur contenue n'ayant pas d'importance. Ce mécanisme permet la synchronisation entre tâches. FreeRTOS propose également des *counting semaphore* et des mutex (sémaphore binaire muni d'un mécanisme d'héritage de priorité).

### 2.2.3 Gestion de la mémoire

Pour chaque nouvelle tâche, sémaphore ou file d'attente il est nécessaire d'allouer de l'espace mémoire en RAM. Les fonctions *malloc()* et *free()* de la bibliothèque standard ne sont pas toujours utilisables. En effet, outre le fait qu'elles ne soient pas toujours disponibles sur le matériel cible, elles prennent un espace de code important, et ne sont pas *thread-safe*. De plus, elles ne sont pas déterministes, ce qui n'est pas envisageable pour un système d'exploitation temps-réel. FreeRTOS propose trois gestionnaires de mémoire différents.
Le premier, le plus simple et le plus léger, ne permet pas de libérer la mémoire précédemment allouée. L'algorithme divise un tableau en blocs alloués au fur et à mesure. Cette solution, déterministe, ne peut être utilisée que si aucune tâche ou file n'est supprimée. La quantité totale de mémoire RAM disponible est configurable.
La deuxième solution proposée utilise un algorithme *best fit* et permet de libérer la mémoire précédemment allouée. Elle peut être utilisée même lorsque des tâches sont supprimées ou créées dynamiquement. Par contre, elle n'est pas déterministe.
La troisième solution est juste un wrapper des fonctions *malloc()* et *free()*. Elle nécessite l'implémentation de la librairie standard et n'est donc pas déterministe. Son utilisation augmente considérablement la taille du noyau.

### 2.2.4 Représentation du temps

FreeRTOS compte le temps en *tick*, un tick correspondant généralement à 1ms. La taille d'un *tick* est configurable (16 ou 32 bits). Sur une architecture 16 bits comme le MSP430, choisir une taille de 32 bits peut se révéler coûteux, en terme d'espace mémoire particulièrement. Une taille 16 bits fait que le temps boucle relativement vite et cela peut se révéler problématique.

## 2.3 Hypothèses et Modèles

Dans cette section nous introduisons des concepts utilisés tout au long du document. Nous discutons de la découverte de voisinage, processus par lequel un nœud prend conscience des nœuds qui l'entourent. Nous discutons également de la modélisation de la couche physique et présentons le modèle réaliste que nous utilisons. Il s'agit du modèle probabiliste du log-normal shadowing qui permet de prendre en compte les irrégularités du medium.

### 2.3.1 Définitions

Nous représentons un réseau de capteurs par un graphe $G = (V, E)$ où $V$ est l'ensemble des nœuds du réseau et $E \subseteq V^2$ l'ensemble des liens de communications. Si un nœud $v$ reçoit les messages envoyés par un nœud $u$, alors $(u, v) \in E$.

Le *voisinage physique* d'un nœud $u$ est défini comme étant l'ensemble des nœuds qui reçoivent ses messages et qui peuvent lui en envoyer. Son *voisinage logique* est constitué des nœuds qu'il considère comme voisins et qui seront donc potentiellement utilisé pour le routage ou par l'application. Nous verrons au chapitre 4 le mécanisme de reconnaissance de voisins implémenté dans Goliath.

### 2.3.2 Découverte de voisinage

La découverte de voisinage se fait généralement par l'envoi de messages HELLO. C'est l'approche que nous adoptons dans nos travaux et qui permet à chaque nœud d'avoir connaissance de son voisinage physique. Chaque nœud envoie régulièrement un message, appelé message HELLO, dans lequel il indique son identifiant et éventuellement la liste des voisins dont il a connaissance si il est nécessaire de connaître son voisinage à deux sauts. L'ensemble des voisins à un saut d'un nœud $u$ est noté $N(u)$ :

$$N(u) = \{v \in V \mid (u, v) \in E\}.$$

Celui de ses voisins à deux sauts est noté $N_2(u)$ :

$$N_2(u) = \{w \mid w \in N(N(u))\backslash N(u) \cup u\}.$$

Lors de la réception d'un message HELLO, le nœud enregistre diverses informations sur le voisin dont il prend connaissance. Il va notamment enregistrer son identifiant, le temps auquel il a été vu pour la dernière fois et éventuellement des informations utiles au routage (distance en nombre de sauts au puits par exemple pour un algorithme de routage tel qu'un gradient). Il peut également, si son chip radio le permet, enregistrer des informations sur la qualité du lien. Les informations généralement fournies sont le RSSI (*Received Signal Strength Indicator*) et le LQI (*Link Quality Indicator*). Le RSSI fournit une information sur la puissance du signal reçu, tandis que le LQI donne une information sur la qualité de ce lien.

L'intervalle d'envoi des messages HELLO est délicat. Un envoi fréquent de messages HELLO permet d'avoir des informations à jour dans la table de voisinages, ce qui important pour des algorithmes (de routage typiquement) utilisant cette table. Cependant, si l'envoi se fait trop

fréquemment, il peut en résulter un gaspillage d'énergie. Une fréquence d'envoi basse permet d'économiser de l'énergie mais peu se révéler très handicapant car le nœud risque de ne pas avoir une bonne connaissance de son voisinage, notamment en cas de mobilité ou d'instabilité des liens, ou de mort de nœuds. Le critère le plus important pour l'envoi des messages HELLO est la mobilité. Une grande mobilité poussera à sélectionner une fréquence d'envoi élevée car le voisinage sera très changeant. Diverses approches existent pour déterminer dynamiquement la fréquence d'envoi à utiliser. On peut notamment citer le protocole AHR (*Adaptive Hello Rate*) qui passe d'une fréquence d'envoi faible à une fréquence élevée et inversement en observant deux métriques (temps de panne des liens et temps sans changement). La solution proposée dans [24] est basée sur la position géographique du nœud et tient compte de la mobilité. Un *beacon* sera envoyé à intervalle de distance parcourue régulier, par exemple à chaque fois qu'un nœud a parcouru $R$ mètres, $R$ étant son rayon de communication. Cette solution ne tient cependant pas compte de la mobilité des autres nœuds. Nous utilisons Les auteurs de [28] proposent le protocole TAP (*Turn-over based Adaptive HELLO protocol*) qui permet d'atteindre la fréquence optimale d'envoi de messages HELLO définie dans [45], notée $f_{hello}$. Celle-ci est définie en fonction de la vitesse relative des nœuds proches ($v_r$) et du rayon de communication :

$$f_{hello} = \frac{v_r}{aR},$$

où $a \in ]0; 1]$ une constant servant à régler la précision désirée. Le protocole TAP propose d'atteindre cette fréquence sans pour autant nécessiter d'informations de positionnement. Il observe le pourcentage de turn-over dans le voisinage du nœud (apparition et disparition de voisins) et adapte la fréquence d'envoi en conséquence.

### 2.3.3 Modélisation de la consommation énergétique

La consommation d'énergie $E(u)$ d'un nœud $u$ lors de l'envoi d'un paquet en utilisant une portée de transmission $r(u)$ est définie comme suit : [19] :

$$E(u) = r(u)^{\alpha} + c.$$

D'autres modélisations, plus complexes, existent. Cependant, nous nous intéressons principalement au coût induit par le choix de la portée, ce qui justifie l'utilisation de ce modèle. Les coûts inhérents aux traitements du paquet sont pris en compte globalement par l'ajout de la constante $c$.

### 2.3.4 Modélisation de la couche physique

La modélisation de la couche physique vise à caractériser le comportement du signal entre un émetteur à un récepteur. Elle peut éventuellement prendre en compte la présence d'autres émetteurs qui créeraient des interférences ou l'impact de l'environnement à modéliser.
Le modèle le plus fréquemment retenu dans la littérature est celui du modèle à seuil que nous présentons dans un premier temps. Ensuite, après avoir mis en évidence les lacunes de ce ce modèle, nous introduisons le modèle (plus) réaliste que nous utiliserons.

#### Modèle à seuil

Dans le modèle à seuil, deux nœuds peuvent communiquer si la distance qui les sépare est inférieure à un seuil, appelé rayon de communication (noté $R$). Lorsque ce rayon de communication est de 1, on parle de modèle du disque unitaire. Plus formellement, dans un modèle à seuil un réseau de capteurs est représenté par un graphe $G = (V, E)$ où $V$ représente l'ensemble des nœuds et

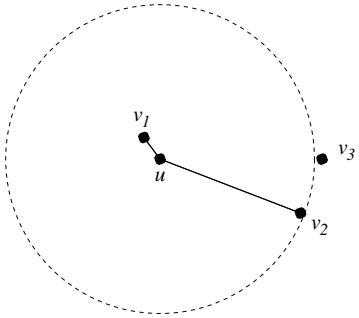



FIGURE 2.6 – *Le modèle du disque unitaire. Le nœud u peut communiquer avec les nœuds $v_1$ et $v_2$, mais aucune communication ne se fera entre u et $v_3$.*

$E \subseteq V^2$ l'ensemble des liens de communication. Le couple $(u, v)$ appartient à $E$ si et seulement si les nœuds $u$ et $v$ peuvent communiquer :

$$E = \{(u, v) \in V^2 \mid d(u, v) \leq R\}.$$

Le graphe sera symétrique et non pondéré si tous les nœuds du réseau utilisent le rayon de communication maximal $R$. Ce modèle est loin de modéliser fidèlement le medium. Il considère que la zone de communication est forcément circulaire, et que la forme du signal est seulement fonction de la distance, et non de l'environnement. Bien que simpliste, ce modèle est cependant utile pour fournir des indications sur comment résoudre certains problèmes, ou pour estimer des propriétés d'un réseau, telle que sa capacité.

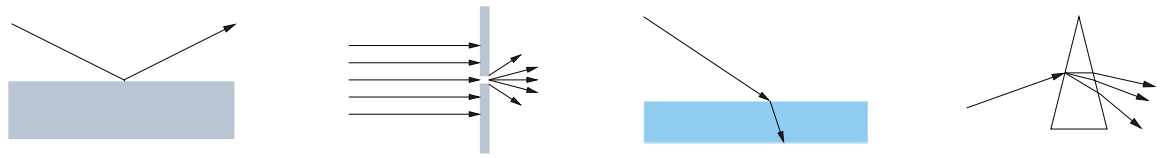
#### Modèles aléatoires

Dans [13] les auteurs présentent une modélisation des réseaux de capteurs utilisant la théorie des graphes aléatoires. Dans ce type de graphes, un arc entre deux nœuds existe avec une probabilité $p$. Dans [18] étudie la probabilité pour un tel graphe d'être connecté, et montre que celle-ci tend vers 1 lorsque le nombre d'arc dépasse un seuil dépendant du nombre de nœuds. Dans [13] les auteurs utilisent la percolation pour générer le graphe aléatoire. Celle ci permet de prendre en compte le fait que seul des nœuds relativement proches seront connectés. Ce modèle prend en compte le fait que les liens de communication ne sont pas uniquement fonction de la distance. Cependant, il considère que les communications se faisant sur un lien ne peuvent jamais échouer.

#### Irrégularité du signal

L'irrégularité du signal peut être dues à deux sortes de facteurs [53] : le matériel et la propagation du signal. Les caractéristiques matériels influant-es incluent le type d'antenne (directionnelle ou omnidirectionnelle), la puissance d'émission utilisée, le gain des antennes, la sensibilité du récepteur, etc. Le niveau de batterie restante au capteur intervient également. Concernant la propagation dans le médium, le premier effet que subit le signal, même en espace libre sans chemins multiples (vue directe entre émetteur et récepteur, aucun obstacle) est celui de l'atténuation. Celle-ci désigne la diminution du signal d'une onde électromagnétique lors de sa propagation. Son effet a tendance à diminuer avec l'augmentation de la bande passante. Dans des conditions idéales, la puissance du signal reçu peut être calculée en fonction de la distance entre l'émetteur et le récepteur par l'équation de Friis :

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L}.$$



(a) réflexion (b) diffraction (c) réfraction (d) dispersion

FIGURE 2.7 – *Phénomènes affectant la propagation d'une onde électromagnétique.*

La puissance du signal ($P_r(d)$) reçu dépend de la puissance du signal émis ($P_t$), du gain linéaire des antennes de l'émetteur ($G_t$) et du récepteur ($G_r$), de la longueur d'onde utilisée ($\lambda$) et le facteur de perte de l'environnement ($L, L \geq 1$). Cependant, l'atténuation que subit un signal dépend également des chemins qu'emprunte l'onde.
Les différents phénomènes observés lors de la propagation de l'onde sont la réflexion, la diffraction, la réfraction et la dispersion. La réflexion se produit lorsque l'onde rencontre une surface réfléchissante. Elle est alors renvoyée avec le même angle que l'angle d'incidence (Figure 2.7(a)). Si la surface n'est pas totalement réfléchissante, elle absorbera une partie de l'onde. La diffraction a lieu lorsque l'onde rencontre un obstacle de l'ordre de grandeur de l'onde, ou des trous. Elle provoque un changement de direction de l'onde (Figure 2.7(b). La réfraction (Figure 2.7(c) lui aussi modifie la direction de l'onde lorsque celle ci arrive dans un milieu de densité différente (milieu aquatique par exemple). La dispersion est l'effet observé lorsque une onde lumineuse rencontre un prisme : les différentes fréquences de l'onde ne se propagent plus à la même vitesse (Figure 2.7(d)). Ces effets sont regroupés sous le terme d'effet de masque ou *shadowing*. Une modélisation prenant en compte cet effet sera de la forme :

$$\text{Puissance reçue} = \text{Puissance émise} - \text{affaiblissement} + \text{effet de masque}.$$

### Le modèle LNS

Le modèle LNS (Log-Normal Shadowing) [40] modélise statistiquement la puissance reçue et prend en compte deux composantes :

- l'**atténuation**, calculée relativement à une distance de référence $d_0$ et un facteur d'atténuation $\beta$,
- la **variation du signal** , par l'utilisation d'une variable aléatoire $X_\sigma$ dont le logarithme suit une loi normale d'écart type $\sigma$.

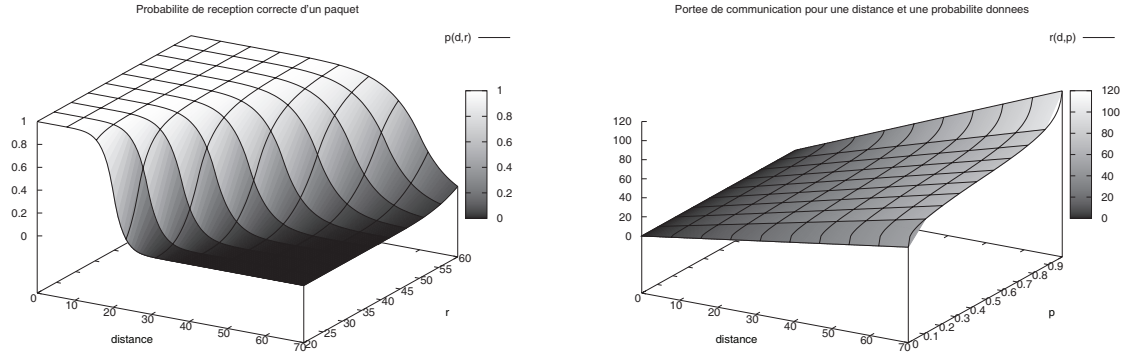
La puissance reçue peut être alors calculée par :

$$\left[\frac{P_r(d)}{P_r(d_0)}\right]_{dB} = -10\beta \log(\frac{d}{d_0}) + X_\sigma.$$

La probabilité qu'un bit soit reçu correctement à une puissance supérieure à $\gamma$ est donnée par :

$$P[P_r(d) > \gamma] = Q\left(\frac{\gamma - P_r(d)}{\sigma}\right).$$

$\gamma$ représente la sensibilité du recepteur. En utilisant cette probabilité pour déterminer la probabilité de réception d'un bit puis d'un paquet, Kuruvila et al. [31] propose une approximation de la fonction de probabilité de réception d'un paquet en fonction de la distance et du rayon de




(a) $p_r(x)$ (b) $r_p(x)$

FIGURE 2.8 – *2.8(a) Probabilité de réception d'un paquet pour une distance et un rayon de communication donnés 2.8(b) Rayon de communication à utiliser pour obtenir une probabilité de réception donnée en fonction de la distance.*

communication $r$ du nœud émetteur. Celui-ci est défini comme étant la distance telle que $p_r(x) = 0.5$. Cette approximation suppose une taille de paquets fixe.

$$p_r(x) = \begin{cases} 1 - \frac{(\frac{x}{r})^{2\beta}}{2} & \text{si} \quad x < r, \\ \frac{(\frac{2r-x}{r})^{2\beta}}{2} & \text{si} \quad r \leq x < 2r, \\ 0 & \text{sinon.} \end{cases}$$

La fonction permettant d'obtenir le rayon à utiliser pour obtenir une probabilité de réception de $p$ entre deux nœuds distants de $x$ est donné par $r_p(x)$. Cette probabilité est donnée par :

$$r_p(x) = \begin{cases} \frac{x}{\sqrt[2\beta]{2(1-p)}} & \text{si} \quad p \in [\frac{1}{2}; 1[, \\ \frac{x}{2 - \sqrt[2\beta]{2p}} & \text{si} \quad p \in [0; \frac{1}{2}[. \end{cases}$$

Il est alors difficile de définir une relation de voisinage. Doit-on considérer un nœud comme voisin dès reception d'un message ? Cette approche n'est pas optimale, en ce que l'on peut recevoir un message d'un nœud lointain, que l'on considérera alors comme voisin, et que potentiellement on utilisera pour le routage alors qu'il n'est pas fiable. Nous détaillons l'approche adoptée au chapitre 4.

## 2.4 Contrôle de topologie

Nous faisons la distinction entre le contrôle de topologie et le contrôle de puissance ou de portée. La portée désigne la distance depuis laquelle un message peut être reçu. La puissance d'émission correspond à la puissance en dBm.

Un algorithme de contrôle de topologie prend en entrée un graphe $G = (V, E), E \subseteq V^2$ et donne en sortie un sous-graphe de $G$. Soit $G_C = (V_C, E_C)$ ce sous-graphe avec $V_C \subseteq V$ et $E_C \subseteq E$. Il peut être obtenu par l'utilisation d'un algorithme de réduction de graphe, l'ordonnancement d'activité, l'établissement de hiérarchies, ou simplement en ignorant certains nœuds. Dans [5] le contrôle de topologie s'effectue par l'ignorance de certains voisins. Chaque nœud conserve les $k$ premiers voisins dont il a connaissance, et ignore les suivants. L'établissement de hiérarchies est l'approche adoptée par les algorithmes basés sur la construction de clusters ou d'ensemble dominants. Un ensemble dominant $D$ est un ensemble de nœuds tels que tout nœud du réseau

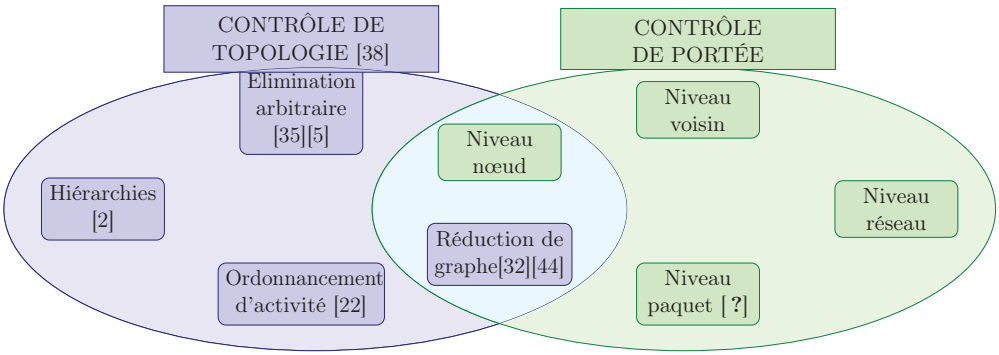



FIGURE 2.9 – *Le contrôle de topologie et le contrôle de portée sont deux problèmes proches. Le contrôle de topologie vise à contrôler le voisinage logique d'un nœud. Le contrôle de puissance de transmission tel que nous le définissons vise à mettre en concordance le voisinage physique et le voisinage logique d'un nœud.*

n'appartenant pas à cet ensemble possède un voisin à un saut qui est dans l'ensemble [25]. Dans ce cas, $V_C = V$. Le clustering consiste à répartir les nœuds dans des ensembles ayant des rôles différents. Un état de l'art des algorithmes de clustering est disponible dans [2]. L'ordonnancement d'activité permet de réduire le nombre de nœuds actifs dans le réseau tout en préservant certaines propriétés intéressantes comme la couverture et la connexité. C'est l'approche adoptée dans [22, 43, 51]. Dans ce cas, $V_c \subset V$. Un algorithme de réduction de graphe conserve tous les nœuds du réseau mais élimine certains liens considérés comme redondants de telles sortes que le graphe résultant satisfasse certaines propriétés intéressantes telles que la planarité ou un degré borné. Un aperçu des différents algorithmes disponibles est fourni dans [38].

### 2.4.1 Élimination arbitraire de voisins

L'élimination arbitraire de voisins est l'approche adoptée par l'algorithme k-neigh présenté dans [5]. Chaque nœud sélectionne ses $k$ plus proches voisins, dans l'optique d'ensuite ajuster sa puissance de transmission au voisin le plus éloigné. Le défi réside dans le choix de $k$. Sa valeur optimale doit être la plus petite possible tout en garantissant la connexité du réseau avec une grande probabilité. Les auteurs de [5] déduise la valeur optimale de $k$ en utilisant les travaux effectués dans [52]. Un nombre de voisins suffisant pour assurer avec une grande probabilité est de l'ordre de $\Theta(n)$, $n$ étant le nombre de nœuds dans le réseau. L'algorithme décrit dans [5] suppose que le nœud possède un moyen d'évaluer la distance entre ses voisins et lui-même. Chaque nœud annonce, à puissance maximale, son identifiant unique et enregistre l'identifiant et la distance estimée de chacun des nœuds qu'il entend. Il sélectionne ensuite les $k$ voisins les plus proches dans cette liste. Il envoie ensuite à puissance maximale la liste des voisins sélectionnés et reçoit celle de ses voisins physiques. Cette phase est nécessaire pour garantir la symétrie des liens. Il ajuste ensuite sa portée de manière à atteindre son voisin logique le plus éloigné. Cette algorithme ne nécessite que l'échange de $2n$ messages pour un réseau de taille $n$.

### 2.4.2 Mise en place de hiérarchies

La mise en place de hiérarchies comprend l'établissement de clusters ou d'ensemble dominants. La plupart de ces algorithmes ont été conçu dans l'optique d'être utilisés pour le routage. C'est le cas par exemple de GAF (*Geographical Adaptive Fidelity*) [51]. Cet algorithme découpe la surface de déploiement en *grille virtuelle* et désigne dans chacune d'elle un nœud actif qui effectuera toutes les opérations de routage pour lesquelles les autres nœuds sont redondants. La taille des grilles virtuelles est choisie telle que chaque nœud dans une grille puisse atteindre tous les nœuds de la grille voisine. Si le rayon de transmission est de $R$, la taille de la grille sera alors de $\frac{R}{\sqrt{5}}$.

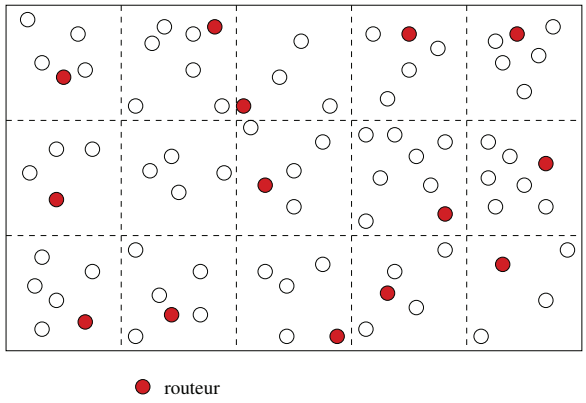



FIGURE 2.10 – *L'algorithme GAF.*

La durée de vie ainsi gagnée est proportionnelle à la densité du réseau. L'inconvénient de cette méthode est qu'elle nécessite un système de positionnement (GPS ou algorithme de localisation), chaque nœud devant être capable de calculer sa position relative par rapport à ses voisins. Le clustering [2] consiste à répartir les nœuds d'un réseau en ensembles disjoints avec pour objectif de réduire la consommation énergétique et favoriser le passage à l'échelle. Dans ces approches, les nœuds sont répartis dans des clusters dans lequel un nœud particulier, le *cluster head* ou *CH* possède des fonctions et un rôle spécifiques. Un algorithme clustering efficace doit veiller à l'équilibrage de charge, c'est-à-dire s'assurer que les clusterheads n'épuisent pas leur énergie prématurément. La tolérance aux pannes est un autre défi. Un algorithme de clustering doit pouvoir assurer la pérennité en cas de défaillance du cluster head. Cette fonction peut être assurée par une reconstruction des clusters, mais cette solution l'inconvénient d'être coûteuse et de nuire à la stabilité. La solution la plus utilisée est de désigner un nœud qui remplacera le cluster head en cas de défaillance. Une rotation de ce rôle entre les nœuds est également envisageable. C'est l'approche adoptée dans LEACH (Low-Energy Adaptive Clustering Hierarchy) [26]. Dans LEACH, la formation des clusters est basée sur la puissance du signal reçu. Les opérations de fusion et d'aggrégation des données sont effectuées au niveau du cluster head. Le temps de convergence de l'algorithme est constant.
Une autre manière de mettre en place des hiérarchies est de construire des ensembles dominants. Un ensemble dominant est un ensemble de nœuds tels que chaque nœud qui n'y appartient possède un voisin dans cet ensemble. L'objectif est alors de trouver un ensemble dominant connecté de taille minimale. Ce problème est NP-Complet. Dans [41], les auteurs présentent une heuristique pour le calcul d'un ensemble dominant de façon distribuée basée sur les travaux de [50]. La notion de nœuds intermédiaires est introduite. Un nœud est intermédiaire si il possède deux voisins qui ne sont pas voisins entre eux. Autrement dit, un nœud est intermédiaire si le graphe formé par ses voisins n'est pas connexe. Tout nœud non intermédiaire ne fait partie de l'ensemble dominant. Un nœud intermédiaire est dominant si le graphe formé par ses voisins de plus haute priorité n'est pas connexe, une plus grande priorité signifiant un degré plus élevé ou un identifiant plus grand. Si ce graphe est connexe le nœud est dominant s'il possède un voisin dont il est le seul voisin.

### 2.4.3 Ordonnancement d'activité

L'ordonnancement d'activité vise à exploiter la densité et la redondance des réseaux de capteurs. Elle est utilisée pour la surveillance de zone. Dans certains scénarios, un nombre important de nœuds est déployé pour effectuer la surveillance d'une zone, plusieurs nœuds peuvent observés la

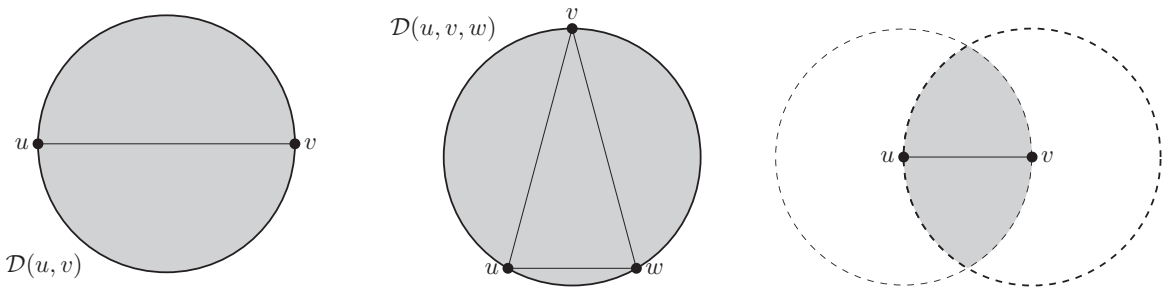
même zone. L'information qu'ils transmettent au puits est alors redondante. L'ordonnancement d'activité consiste à alterner les phases d'activité des capteurs de manière à économiser de l'énergie tout en garantissant la fonctionnalité du réseau, c'est-à-dire en s'assurant que la surface couverte reste identique. La sélection des nœuds devant rester actif peut se faire par la construction d'ensembles dominants de surface [7]. Un ensemble dominant de surface est un ensemble de nœuds tel que tout point dans la zone de surveillance soit couvert par un nœud appartenant à l'ensemble. L'objectif est de trouver un tel ensemble de taille minimale et connexe. La couverture peut être simple ou multiple. Un algorithme de couverture simple assure que tout point de la zone soit couvert par au moins un capteur. Une algorithme de $k$-couverture assure la couverture de tout point par au moins $k$ capteurs lorsque cela est possible. Une approche centralisée nécessite la présence d'une entité ayant connaissance de la totalité du réseau, empêchant le passage à l'échelle ou le déploiement sans infrastructure. Un approche localisée, plus appropriée, ne repose sur aucune infrastructure et permet de faciliter plus facilement les changements de topologie, même dans un réseau à large échelle. Généralement, un tel algorithme commence par une phase de découverte de voisinage. Chaque nœud prend alors la décision de rester actif ou inactif, et en informe ses voisins par l'envoi d'un message. L'envoi peut se faire soit lorsqu'il décide de rester actif, soit il décide de passer en mode inactif.

### 2.4.4 Réduction de graphe

Les algorithmes utilisant la réduction de graphes cherchent à éliminer les liens redondants en utilisant la représentation géométrique du réseau. Dans cette sous-section nous présentons dans un premier temps les propriétés intéressantes que peuvent avoir un algorithme de réduction de graphe. Nous nous intéressons ensuite à différentes approches en mettant en avant leurs avantages et leurs inconvénients.

#### Évaluation de réduction

*Connexité :* Il s'agit du critère fondamental. Un algorithme de réduction de graphe doit préserver la connexité, c'est-à-dire que si le graphe en entrée est connexe, le graphe obtenu après réduction est lui aussi connexe.
*Symétrie :* La symétrie suppose que si un lien de $u$ vers $v$ est conservé, alors le lien de $v$ vers $u$ le sera également. Il est important de pouvoir conserver la symétrie étant donné qu'elle est une des hypothèses de départ de nombreux algorithmes (de routage par exemple). La symétrie peut être obtenue à la suite de la réduction par échange de messages si l'algorithme utilisée ne la garantit pas.
*Planarité :* Un graphe est dit planaire si aucune arête (ou arc) n'en croise une autre. La planarité est une propriété intéressante pour certains algorithmes de routage, notamment des algorithmes de routage géographique.
*Degré moyen :* Le degré moyen d'un nœud est important pour le passage à l'échelle. Un degré borné et relativement faible est idéal. Il facilite notamment les opérations de routage.
*Interférences :* Le contrôle de topologie a notamment pour objectif de réduire les interférences. La baisse du degré moyen ne garantit la baisse des interférences.
*Facteur d'étirement :* Le facteur d'étirement (*stretch factor*) désigne le facteur par lequel est multipliée la distance en nombre de sauts entre deux nœuds. Si les $u$ et $v$ était à un saut dans le graphe original, et qu'ils sont à une distance $x$ sauts dans le graphe réduit, le facteur d'étirement est de $x$. Un facteur d'étirement faible est préférable.
*Informations nécessaires :* Étant donné le côté contraint des capteurs, les informations nécessaires à la réduction du graphe sont un paramètre important dans le choix d'un algorithme de réduction de graphe. Un algorithme localisé est préférable. Un algorithme est dit localisé s'il peut être




(a) Graphe de Gabriel (b) Graphe de Delaunay (c) Graphe des plus proches voisins

FIGURE 2.11 – *Réduction de graphes : dans les trois approches, un lien est conservé s'il n'existe aucun autre nœud dans la zone grisée.*

effectué avec une connaissance du voisinage à $k$ sauts, $k$ constant par rapport à la taille du réseau [39].

Un algorithme de réduction de graphe idéal aura donc un facteur d'étirement faible, un nombre linéaire d'arcs. Il devra résulter en un degré borné, tout en pouvant être distribué et local.

### Le graphe de Gabriel

Le graphe de Gabriel [21] d'un graphe $G = (V, E)$ est défini comme étant le sous-graphe de $G$ qui relie les points de $V$. Soit $u$ et $v$ deux nœuds de $V$ et $\mathcal{D}(u, v)$ le disque de diamètre $d(u, v)$ déterminé par les points $u$ et $v$. Le lien entre $u$ et $v$ appartient au graphe de Gabriel de $G$ si et seulement si $\mathcal{D}(u, v)$ n'inclut aucun autre point de $V$.

### Triangulation de Delaunay

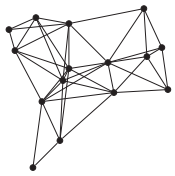
Les liens entre trois points $u$,$v$ et $w$ sont conservés s'il n'existe aucun autre point dans le cercle circonscrit du triangle $(u, v, w)$.

### Approches basées sur des cônes

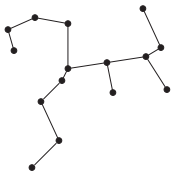
Le graphe de Yao est un exemple de graphe basé sur l'utilisation de cônes. Il s'agit d'un graphe paramétré par une constante $k \geq 6$. Chaque nœud découpe sa zone de communication en $k$ cônes d'angles égaux. Il conserve un seul voisin par cône. Le graphe obtenu est orienté et faiblement connecté (ce qui signifie que son équivalent non orienté est connexe). Un autre algorithme basé sur les cônes, CBTC (*Cone-Based Topology Control*) est présenté dans [47]. Les voisins sont parcourus par distance croissante. Un lien avec un voisin est maintenu si l'angle qu'il couvre (déterminé par un paramètre $\alpha$) n'est pas encore couvert. Le graphe obtenu peut ne pas être symétrique.

### Arbre recouvrant mininal

L'arbre recouvrant minimal d'un graphe $G$ est l'arbre $T$ qui contient tous les sommets de $G$ tout en minimisant la somme des poids des arêtes. Il est couramment utilisé pour la diffusion (par exemple dans [49]) car il garantit que tous les nœuds du réseau soient atteints. Dans ces approches, le poids des arêtes représente la distance entre les nœuds.Comme il s'agit d'un arbre, il existe un chemin et un seul entre chaque pair de nœuds. L'algorithme généralement utilisé pour calculer cet arbre est l'algorithme de Prim [37] qui consiste à construire un arbre à partir d'un point arbitraire du graphe. Dans une utilisation dans un réseau de capteurs, le nœud de départ est généralement le puits. Comme cet algorithme est centralisé, son utilisation est coûteuse, le puits devant posséder



(a) UDG ($\delta$ = 5.6)



(b) MST ($\delta$ = 1.87)

(c) RNG ($\delta$ = 2.34)

(d) LMST ($\delta = 2$)

FIGURE 2.12 – *Illustration réduction de graphe*

une connaissance totale du réseau et transmettre aux nœuds les liens à conserver. L'algorithme de Prim conserve la connexité du réseau. Les auteurs de [32] propose une variante locale de cet algorithme appelée arbre recouvrant minimal local ou LMST (*Local Minimum Spanning Tree*). Il s'agit en fait pour chaque nœud de calculer l'arbre recouvrant minimal sur son voisinage, ce qui nécessite la connaissance de la position de ces nœuds ou du voisinage à deux sauts. Le graphe obtenu (qui n'est pas un arbre) n'est pas symétrique, rendant nécessaire l'échange de messages pour « symétriser » le graphe.

**Le graphe des plus proches voisins**

Le graphes des plus proches voisins, appelé également graphe de voisinage relatif (*Relative Neighborhood Graph* ou RNG), a été décrit par Toussaint [44]. Comme son nom l'indique seuls les voisins les plus proches sont conservés. Lorsqu'on rencontre un triangle, le lien le plus « long » est éliminé. Soit $RNG(G) = (V, E_{RNG})$ le sous-graphe des plus proches voisins de $G = (V, E)$. L'ensemble des liens $E_{RNG}$ est défini comme suit :

$$E_{RNG} = \{(u, v) \in E \mid \forall w \max d(u, w), d(v, w) \geq d(u, v)\}.$$

Le graphe RNG d'un graphe connexe est également connexe. Le degré est au maximum de 6. Un algorithme similaire est présenté dans [48]. Chaque nœud ordonne ses voisins selon un critère quelconque (en utilisant la distance, on retombe sur le RNG) et utilise le même algorithme que dans le graphe de voisinage relatif. Le graphe des plus proches voisins est un sous-graphe du graphe de Gabriel.

## 2.5 Ajustement de portée

Le contrôle de la puissance d'émission a pour but de rendre les liens existant plus efficaces et moins coûteux en terme de consommation énergétique. Nous reprenons la taxonomie introduite dans [33]. Le contrôle de la puissance d'émission peut se faire à quatre niveaux de granularité : au niveau du réseau, du nœud, de chaque voisin ou de chaque paquet. Les solutions appliquées au niveau réseau ont pour objectif de trouver la portée de transmission adéquate pour s'assurer qu'un réseau déployé aléatoirement ait une probabilité élevée d'être connexe. Lorsque l'ajustement de portée s'effectue au niveau du nœud, le but est d'assigner à chaque nœud du réseau une portée optimale qui minimisera les interférences, et/ou la consommation énergétique tout en préservant la connexité. Le choix d'une portée pour chaque voisin permet au nœud de choisir une portée de transmission différente selon le voisin avec lequel il souhaite communiquer. Dans le même esprit, il est possible de choisir une puissance d'émission différente pour chaque paquet envoyé. C'est l'approche adoptée dans [33]. Nous nous intéressons au contrôle de portée de transmission au niveau du nœud. Un bon ajustement de portée rapprochera le degré physique d'un nœud de son degré logique. En assurant cette fonction, il permet de diminuer les interférences et permet

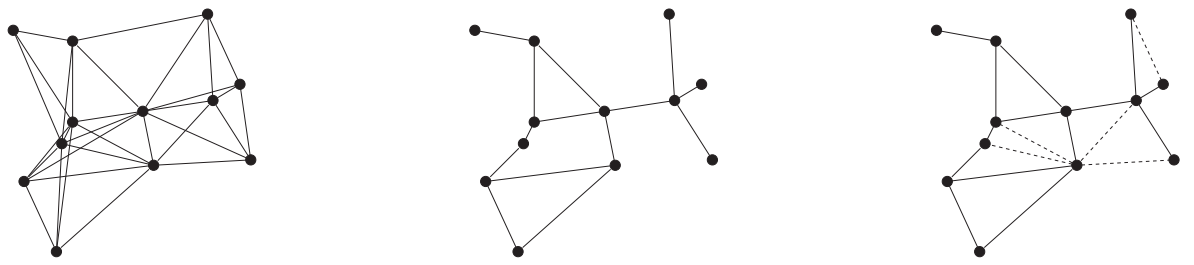
d'économiser de l'énergie. Celle-ci se fait à deux niveaux principalement : au niveau de l'émetteur (qui dépensera moins d'énergie pour l'émission de son paquet), des nœuds qui ne sont plus à portée suite à l'ajustement de portée (ils ne devront plus traiter des paquets qui ne leur sont pas destinés).

La contrôle de topologie couplé au contrôle de puissance de transmission (au niveau du nœud) correspond au problème de l'assignement de portée défini dans le cadre de l'utilisation du modèle à seuil comme suit. On considère un ensemble de nœuds $V$ répartis dans un espace bidimensionnel. Soit $G = (V, E)$ le graphe obtenu lorsque tous les nœuds utilise la puissance de transmission maximale $R$, $d$ une fonction de distance de $E$ dans $\mathbb{R}^+$. Un assignement de portée $r$ est une fonction de $V$ in $\mathbb{R}$. Le graphe résultant d'un assignement de portée un graphe orienté noté $G_r = (V, E_r)$. Son coût $C(r)$ est donné par :

$$C(r) = \sum_{u \in V} c(u)$$

où $c$ est une fonction de $V$ dans $\mathbb{R}$ donnant le coût pour un nœud d'utiliser la puissance de transmission qui lui a été assignée.
Un assignement de portée est dit *valide* si et seulement si il préserve la connexité du réseau : un réseau connexe lorsque les nœuds ont une portée de $R$ doit le rester lorsque la fonction d'assignement de portée est utilisée. Il a été démontré dans [30] que ce problème est NP-dur. De plus, ce problème et la validité d'une solution n'ont été définis que dans le cadre de l'utilisation du modèle à seuil. En effet, la notion de connexité utilisée pour juger de la validité d'un assignement de portée n'est pas directement transposable dans le cas de l'utilisation d'un modèle de couche physique réaliste. Pour remédier à ce problème nous définissons, dans le chapitre 3, la notion de $\lambda$-proximité.

Parmi les solutions possibles au problème de l'assignement de portée, nous étudions plus particulièrement celles se basant sur une réduction de graphe. Le graphe obtenu à puissance maximale est dans un premier temps réduit en utilisant une des approches présentées précédemment. Le graphe obtenu est noté $G_{GR}$. Chaque nœud se voit ensuite assigné la puissance de transmission minimale lui permettant d'atteindre son voisin le plus éloigné dans $G_{GR}$. Ce processus est illustré sur la figure 2.13. Nous verrons dans le chapitre 3 comment adapter les réductions de graphe sélectionnées au cas d'une couche physique réaliste.



(a) UDG(G) (b) $G_{GR}$ (c) $G^+_{GR}$

FIGURE 2.13 – *Contrôle de topologie par réduction de graphe. Le graphe initial 2.13(a) est obtenu par l'utilisation de la puissance d'émission maximale R. La figure 2.13(b) présente le sous-graphe après l'application d'un algorithme de réduction de graphe. Chaque nœud ajuste sa puissance d'émission de manière à atteindre son voisin le plus éloigné dans ce sous-graphe $G_{GR}$. Le graphe $G^+_{GR}$ ainsi obtenu contient $G_{GR}$. Les liens en pointillés dans 2.13(c) représente les liens créés par cette assignement de portée qui n'étaient pas dans $G_{GR}$.*

## 2.6 Conclusion

Dans ce chapitre, nous avons présenter différentes piles protocolaires pour réseaux de capteurs ainsi que les systèmes d'exploitation qui leur sont couramment associés. Dans un souci de pouvoir mettre en évidence l'impact des solutions de contrôle de topologie que nous avons implémentées, nous avons développer notre propre pile de communication, Goliath, décrite dans le chapitre 4 Nous avons également décrit plusieurs stratégies de contrôle de topologie envisageables dans un tel réseau. Notre étude se focalise particulièrement sur le contrôle de topologie basé sur une réduction de graphe. L'objectif étant de rapprocher la théorie du contrôle de topologie et son implémentation réelle, nous utilisons un modèle de couche physique réaliste. Le modèle retenu est le modèle du log-normal shadowing, mais nos travaux sont valables pour tout modèle de couche physique probabiliste.

# Contrôle de topologie et couche physique réaliste

# 3

« Well if there's a message I must bring to
you,
Love all and trust a few. »

**Groundation**,
*Throwing Stones - Each one teach one.*

Sommaire



Nous discutons dans ce chapitre de la notion de $\lambda$-proximité, introduite pour juger de la qualité d'une topologie lors de l'utilisation d'une couche physique réaliste. Nous illustrons ce concept avec l'utilisation du modèle LNS introduit précédemment. Nous présentons ensuite une méthode

d'adaptation des algorithmes de réduction de graphe dans le cadre d'une utilisation avec un modèle probabiliste. Cette étude nous mène à présenter un exemple de réduction de graphe utilisant comme métriques la somme des logarithmes des probabilité. Nous nous intéressons ensuite au problème de l'assignement de portée lors de l'utilisation d'une couche physique réaliste. Après avoir présentés des algorithmes d'assignement de portée conservant la $\lambda$-proximité, nous montrons qu'il n'existe pas d'algorithme localisé permettant d'assurer cette préservation. Nous proposons donc la notion de $\lambda$-proximité localisée et présentons des algorithmes localisés la conservant.

## 3.1 Notion de λ-proximité

### 3.1.1 Définition

Nous introduisons dans ce chapitre la notion de λ-proximité [29]. Nous considérons un graphe $G = (V, E)$ dans lequel chaque arc $(u, v)$ est muni d'un poids correspondant à la probabilité qu'a $v$ de recevoir correctement un message envoyé par $v$. Nous ne faisons pas d'hypothèses quant à la manière de calculer cette probabilité. Un chemin entre les nœuds $a_1$ et $a_k$ est une succession de nœuds $s_{a_1,a_k} = a_1, a_2, \ldots, a_k \in V$. La probabilité qu'un message soit transmis correctement de $u$ à $v$ en utilisant le chemin $s$ est notée $P(s_{a_1,a_k})$. Le message doit parcourir tous les nœuds du chemin dans l'ordre présenté dans $s$. Cette contrainte est justifiée par l'utilisation de protocoles de routage dans les réseaux de capteurs, notamment si ce dernier suppose que le nœud routant un paquet en choisi le prochain relais. $P(s)$ est donnée par :

$$P(s_{u,v}) = p(u, a_1) \times p(a_1, a_2) \times \cdots \times p(a_{k-1}, a_k) \times p(a_k, v).$$

L'ensemble des chemins de $u$ à $v$ dans $G$ tels que $P(s_{u,v}) \geq \lambda$ est noté $\mathcal{C}_G(u, v)$. La λ-proximité est définie comme suit :

**Definition 3.1.** *Un réseau est dit* λ-proche *si et seulement s'il existe entre chaque nœud du réseau un chemin dont la probabilité de communication correcte est d'au moins* λ.

Un réseau est donc λ-proche si et seulement si :

$$\forall u, v \in V \quad \exists s_{u,v} \quad P(s_{u,v}) \geq \lambda.$$

Cette définition permet de s'assurer qu'il existe une même probabilité de communication correcte entre chaque nœud du réseau. Elle rend également possible la comparaison de différentes topologies d'un point de vue qualitatif.

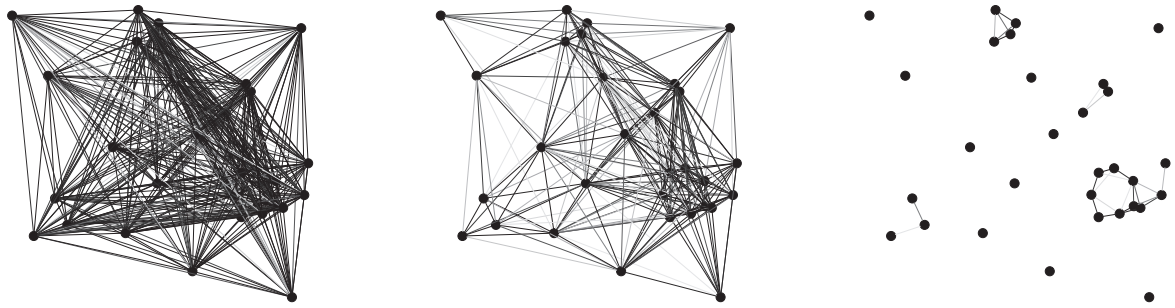
### 3.1.2 λ-proximité et modèle LNS

Dans nos travaux, nous utilisons à des fins d'illustration le modèle LNS, présenté dans le chapitre 2, comme modèle de couche physique probabiliste. Cependant, nos travaux sont applicables à d'autres modèles probabilistes. Dans le modèle LNS, la probabilité de réception d'un paquet est fonction de la distance parcourue $x$ :

$$p_r(x) = \begin{cases} 1 - \frac{(\frac{x}{r})^{2\beta}}{2} & \text{si} \quad x < r, \\ \frac{(\frac{2r-x}{r})^{2\beta}}{2} & \text{si} \quad r \leq x < 2r, \\ 0 & \text{sinon.} \end{cases}$$

La figure 3.1 montre l'exemple d'un même réseau composé de 30 nœuds avec l'utilisation de rayons de communication différents. La surface utilisée est de taille $10 \times 10$. La clarté d'un lien dénote une faible probabilité de communication correcte. Un rayon de communication de 10 donne un graphe quasi-complet.
La figure 3.2 donne la λ-proximité moyenne maximale que l'on peut espérer atteindre en fonction de la densité par zone de communication. On peut observer que cette λ-proximité tend vers 1 lorsque la densité augmente.

Les modèles probabilistes soulèvent également une autre interrogation qui est de savoir quels nœuds doivent être considérés comme voisins. Nous introduisons un seuil, noté $\sigma$, au delà duquel un nœud est considéré comme voisin. Deux nœuds sont voisins si la probabilité de communication correcte directe entre eux est supérieure ou égale à $\sigma$. Si $\sigma = 0.5$, la topologie du graphe obtenue est celle obtenue dans le modèle UDG, étant donné que le rayon de communication $R$ est défini comme étant la distance telle que $p(R) = 0.5$.



(a) $R = 10$ (b) $R = 5$ (c) $R = 1$

FIGURE 3.1 – *Topologie du réseau en utilisant le modèle LNS avec différents rayons de communication. La clarté d'un lien augmente lorsque la probabilité de communication correcte diminue.*

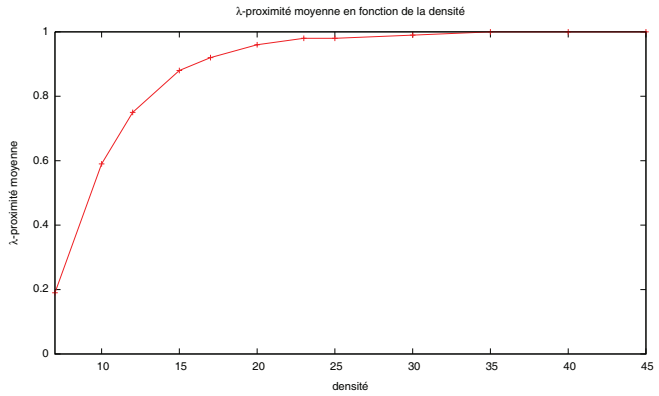



FIGURE 3.2 – *λ-proximité moyenne. Celle-ci correspond à la λ-proximité que l'on peut atteindre en moyenne dans un réseau d'une densité donnée. Les moyennes sont effectuées sur 1000 réseaux de 100 nœuds dont les positions sont aléatoires.*

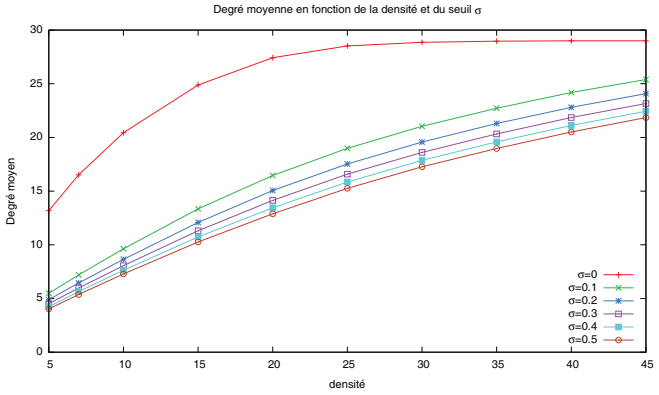



FIGURE 3.3 – *Degré moyen en fonction du seuil de voisinage σ pour différentes densités.*

## 3.2 Réduction de graphe

Dans cette section nous nous intéressons aux réductions de graphe préservant la $\lambda$-proximité. Nous présentons ensuite un algorithme de contrôle de topologie applicable lors de l'utilisation de métrique de qualité de service. Cet algorithme peut être utilisé pour la sélection des relais dans un protocole comme OLSR dans lequel un nœud sélectionne un sous ensemble de ses voisins lui servant de relais pour les messages diffusés dans tous le réseau.

### 3.2.1 Réduction de graphe et $\lambda$-proximité

Nous présentons dans cette section comment adapter un algorithme de réduction de graphe défini dans le modèle UDG pour qu'il soit utilisable avec un modèle de couche physique probabiliste. Soit $G' = (V, E')$ le graphe obtenu après réduction. Préserver la $\lambda$-proximité dans ce graphe signifie que s'il existait un chemin $s$ entre les nœuds $u$ et $v$ tel que $P(s) \geq \lambda$, alors il existe un chemin $s'$ entre ces mêmes nœuds dans le graphe réduit tel que $P(s') \geq \lambda$. Plus formellement un algorithme de réduction de graphe doit satisfaire la condition suivante :

$$\forall u, v \in V(\exists s = u, \ldots, v \mid P(s) \geq \lambda) \Rightarrow (\exists s' = u, \ldots, v \mid (u, b_0), \ldots, (b_k, v) \in E' \wedge P(s') \geq \lambda).$$

L'objectif d'une réduction de graphe est d'éliminer certains liens tout en conservant les plus avantageux. Dans notre cas, les liens les plus avantageux sont ceux qui maximisent la probabilité de réception. Maximiser la probabilité d'un chemin $s$ revient à maximiser :

$$\begin{aligned} \max(\prod_{i=0}^{n-1} p(a_i, a_{i+1})) &\Leftrightarrow \max(\sum_{i=0}^{n-1} \ln P(a_i, a_{i+1})) \\ &\Leftrightarrow \min(\sum_{i=0}^{n-1} -\ln p(a_i, a_{i+1})) \\ &\Leftrightarrow \min(\sum_{i=0}^{n-1} \ln \frac{1}{p(a_i, a_{i+1})}). \end{aligned}$$

Nous obtenons une métrique additive qui permet d'utiliser des réductions de graphe classiques comme celles présentée dans le chapitre 2, section 2.4.4. Nous choisissons d'utiliser le graphe de voisinage relatif, car il permet d'éliminer les liens les moins fiables. En supposant que, comme c'est le cas avec le modèle LNS, la probabilité de réception dépend de la distance, cet algorithme éliminera les liens les plus longs. De plus, il s'agit d'un algorithme localisé nécessitant peu d'informations (le voisinage à deux sauts) et qui conserve la symétrie des liens. L'approche que nous proposons peut être appliquée à d'autres algorithmes de réduction de graphe. Dans le RNG, un lien $(u, v)$ sera conservé par $u$ si il n'existe pas de liens alternatifs en deux sauts qui permet d'atteindre $v$ avec une probabilité égale ou supérieure :

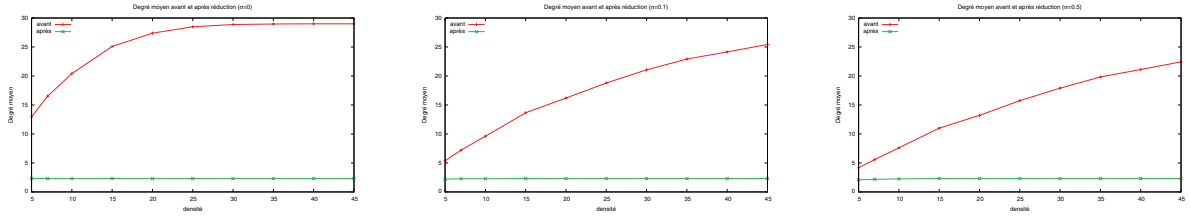
$$(u, v) \in E' \Leftrightarrow p(u, v) \geq \max(p(u, a_i) \times p(a_i, v)).$$

### 3.2.2 Réduction de graphe, routage et $\lambda$-proximité

Lors du routage d'un paquet, il peut être nécessaire de s'assurer de la qualité du chemin emprunté en terme probabilité de succès de la transmission. Nous présentons dans cette sous-section un algorithme de sélection de voisins préservant la $\lambda$-proximité.
Nous nous plaçons dans le cas de l'utilisation d'un protocole de routage pro-actif tel que OLSR [10] et sa variante avec prise en compte de la qualité de service QOLSR [3]. OLSR (*Optimized Link*

**Input** : $N(u), N_2(u)$
**Output** : $N'(u)$ voisinage conservé par le nœud $u$
$N'(u) \leftarrow N(u)$ ;
**foreach** $v \in N(u)$ **do**
  **if** $\exists w \in N(u) \mid (P(u,w) \times P(w,v)) > P(u,v)$ **then**
    $N'(u) \leftarrow N(u) \backslash \{v\}$ ;
  **end**
**end**

Algorithme 1 – *Algorithme de réduction de graphe se basant sur le graphe de voisinage relatif.*




(a) $\sigma = 0$ (b) $\sigma = 0.1$ (c) $\sigma = 0.5$

FIGURE 3.4 – *Degré obtenu avant et après réduction du graphe avec différentes valeurs de seuil au delà duquel un nœud est considéré comme voisin. Les résultats sont obtenus sur 1000 graphes de 100 nœuds dont la position est générée aléatoirement.*

*State Routing*) permet de réduire la charge du réseau en limitant le nombre de nœuds impliqués lors d'un broadcast ou de la dissémination d'informations relatives à la topologie (utilisées pour le routage) par la sélection de nœuds chargés de ces tâches, les MPR (*Multi-point relay*). Chaque nœud sélectionne un sous ensemble de ses voisins à un saut comme MPR. Ce sous-ensemble doit permettre d'atteindre tous les voisins à deux sauts du nœud. L'algorithme de sélection des MPR présenté dans OLSR se fait en deux phases. La première consiste à sélectionner les MPR pour les voisins à deux sauts isolés, c'est-à-dire couverts par un seul voisin à un saut. Durant la deuxième phase, le nœud sélectionne les MPR nécessaires pour couvrir les voisins restants, en sélectionnant en priorité le voisin à un saut couvrant le plus grand nombre de voisins à deux sauts non couverts. Chaque nœud diffuse ensuite dans ses messages HELLO les nœuds qu'il a sélectionné comme MPR. Chaque nœud ayant été sélectionné comme MPR diffuse dans tous le réseau des messages de contrôle de topologie contenant la liste de ses sélecteurs MPR (nœuds qui l'ont sélectionné comme MPR). Cette heuristique vise à minimiser la taille de l'ensemble des MPR et suppose que tous les liens sont équivalents (il n'y a pas de lien meilleur qu'un autre). Cette approche n'est pas la plus appropriée pour garantir la qualité de service puisque ces liens peuvent ne pas satisfaire les contraintes imposées.
Une version d'OLSR prenant en compte la qualité de service est proposée dans [3]. Il s'agit du protocole QOLSR (*QoS OLSR*) qui modifie l'heuristique de sélection des MPR. Deux versions de cette heuristique sont proposées, modifiant la seconde phase de l'heuristique originale. La première version, MPR-1, choisit le nœud avec la plus grande bande passante en cas d'égalité concernant le nombre de nœud couvert. La seconde (MPR-2), plus performante, ne prend pas en compte le nombre de nœuds couverts lors du choix d'un MPR dans la seconde phase mais la bande passante associée au lien. Ces versions ont un inconvénient majeur : elles ne modifient que la seconde phase de l'heuristique. Or, il a été montré dans [6] que 75% des MPR sont sélectionnés dans la première phase. Les versions proposées sélectionnent un ensemble ne différant pas beaucoup de

celui sélectionné par l'heuristique originale. L'utilisation d'une de ces heuristiques pour conserver la $\lambda$-proximité ne se révélerait donc pas beaucoup plus performante.
Dans [34] les auteurs proposent de dissocier la sélection des MPR et la sélection des relais utilisés pour satisfaire des contraintes de qualités de service. L'ensemble des MPR (sélectionné avec l'heuristique originale) est donc toujours utilisé pour la diffusion des messages de contrôle de topologie. Le second ensemble, appelé QANS *QoS Advertised Neighbor Set* est utilisé pour le routage des paquets avec QoS. Sa sélection est basée sur le filtrage de topologie : le nœud exécute un algorithme de réduction de graphe (en l'occurrence, le RNG) dans son voisinage à deux sauts et sélectionne les nœuds qui maximisent (resp. minimisent) la bande passante (resp. le délai) pour un nœud à deux sauts dans le graphe réduit. Il est ainsi possible de sélectionner un chemin de deux sauts pour atteindre un voisin à un saut s'il se révèle meilleur en terme de QoS. Dans notre cas, il sélectionnerait celui qui minimise le logarithme de la probabilité sur les liens.
Nous adoptons la même approche : nous sélectionnons, indépendamment de l'ensemble des MPR, un ensemble utilisé pour le routage avec préservation de la $\lambda$-proximité et annoncé dans les messages HELLO. L'originalité de notre approche repose sur le fait que nous autorisons un nœud à sélectionner des chemins de longueurs $k$,$k \geq 2$, pour atteindre un voisin à un ou deux sauts. L'ensemble annoncé sera l'ensemble des premiers nœuds de ces chemins. Les simulations montrent que notre solution - *First Node on Best Path (FNBP)* - donnent des routes proches de l'optimal calculé de manière centralisé. Pour obtenir une métrique additive, nous utilisons les logarithmes des inverses des probabilités associées aux liens, l'objectif étant de minimiser leur somme.

### Notations

On note $ANS(u)$ l'ensemble de nœuds sélectionnés par $u$. Le logarithme de l'inverse de la probabilité de communication correcte entre les nœuds $u$ et $v$ est notée $\mathcal{L}(u,v)$. Soit $p(x_0,x_n) = x_0x_1 \dots x_ix_{i+1} \dots x_n, \forall i\ x_i \in V, (x_i,x_{i+1}) \in E$ un chemin entre $x_0$ et $x_n$ dans $G$. Le logarithme de la probabilité associé à ce chemin est noté $\mathcal{L}(p)$ et est calculé de la manière suivante :

$$\mathcal{L}(p) \quad = \quad \sum_{0}^{n-1} \mathcal{L}(x_i, x_i+1).$$

Soit $\mathcal{P}_{\mathcal{L}}(u,v)$ l'ensemble des chemins de $u$ à $v$ qui minimisent la somme des logarithmes des inverses des probabilités et $\widetilde{\mathcal{L}}(u,v)$ la valeur de cette somme *i.e.* $\widetilde{\mathcal{L}}(u,v) = \{\mathcal{L}(p) \mid p \in \mathcal{P}_{\mathcal{L}}(u,v)\}$. L'ensemble des nœuds $w$ tels que $w$ est le premier nœud du chemin $p$ pour tout $p$ dans $\mathcal{P}_{\mathcal{L}}(u,v)$ est noté $f\mathcal{P}_{\mathcal{L}}(u,v)$.
Pour chaque nœud $u$ nous définissons l'opérateur de comparaison binaire $\prec_L^u$ tel que $w \prec_L^u v, \forall u,v,w \in V, v,w \in N(u)$ si et seulement si $\mathcal{L}(u,w) < \mathcal{L}(u,v)$ ou $\mathcal{L}(u,w) = \mathcal{L}(u,v) \wedge (id_v < id_w)$. $\min_{\prec_L}$ est la fonction minimum associée à cet opérateur. Un exemple de réseau est illustré sur la figure 3.5. Seul le voisinage à deux sauts du nœuds $u$, ainsi que les liens de ce voisinage dont il n'a pas connaissance (en pointillés) sont représentés. Dans cet exemple, $\mathcal{P}_{\mathcal{L}}(u,v_5) = \{uv-1v_5, uv_2v_5\}$ et $\mathcal{P}_{\mathcal{L}}(u,v_3) = \{uv_2v_3\}$. La valeur des meilleurs chemins entre $u$ et $v_5$ est $\widetilde{\mathcal{L}}(u,v_5) = 5$. Nous avons aussi $\widetilde{\mathcal{L}}(u,v_3) = 5$. L'ensemble des premiers chemins sur les chemins optimaux entre $u$ et $v_5$ est l'ensemble $f\mathcal{P}_{\mathcal{L}}(u,v_5) = \{v_2,v_1\}$. Cet ensemble est $f\mathcal{P}_{\mathcal{L}}(u,v_3) = \{v_2\}$ pour $u$ et $v_4$. Pour le nœud $u$ on a $v_1 \prec_L^u v_7$ comme $\mathcal{L}(u,v_1) < \mathcal{L}(u,v_7)$. Comme le logarithme de l'inverse des probabilités entre $u$ et $v_2$ est la même que celle entre $u$ et $v_6$, $v_2 \prec_L^u v_6$ comme $v_2$ possède un identifiant plus petit que $v_6$.

### Algorithme

Dans la proposition originale d'OLSR [10], les MPR sont utilisés dans deux objectifs : la diffusion des messages et des messages de contrôle de topologie servant à établir les tables de routage.

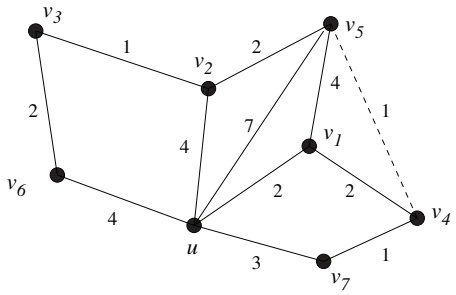



FIGURE 3.5 – *Exemple de réseau. Le poids sur les liens sont les logarithmes des probabilités. Seul le voisinage à deux sauts de u est montré. Le lien en pointillé représente un lien existant mais dont u n'a pas connaissance.*

La sélection des MPR se fait de telle sort que le chemin sélectionné soit le plus court en terme de nombre de sauts. Bien qu'il s'agisse de la stratégie adéquate pour le broadcast, elle peut se révéler moins appropriée pour le routage avec contraintes associées aux liens [34]. Nous adoptons l'approche proposée dans [34] qui consiste à choisir des ensembles ANS et MPR distincts. L'idée sous-jacente de cette proposition est qu'un lien direct entre deux nœuds peut ne pas être optimal en terme de probabilité de communication correcte. Pour obtenir de meilleures performances, nous autorisons un nœud à sélectionner un chemin de $k$ sauts ($k \geq 2$) au lieu d'un lien direct s'il permet d'obtenir de meilleures performances. Par exemple, sur le réseau de la figure 3.5, $u$ a alors la possibilité de choisir le chemin $uv_2v_5$ pour atteindre $v_5$, obtenant ainsi une somme de 6. L'utilisation du lien direct, comme imposée dans la sélection des MPRs, ne lui aurait permis d'obtenir une somme d'au minimum 7.
Fournir un chemin optimal en terme de probabilité de communication correcte entre deux nœuds quelconques du réseau ne peut être fait qu'en utilisant un algorithme centralisé. EXEMPLE Comme le montre la figure 3.5, un algorithme localisé ne peut y parvenir. Un algorithme localisé devrait être capable de prendre des décisions basées sur son voisinage à $k$ sauts, $k$ étant constant. Dans notre exemple où $k = 2$, le nœud n'a pas connaissance de l'existence du lien $(v_4, v_5)$. Il choisira le chemin $uv_1v_5$ avec une somme de 6 pour atteindre $v_9$ alors que le chemin $uv_1v_4v_5$ donne une somme de 5. Cependant, l'algorithme que nous proposons fournit, de manière localisée, le chemin optimal en terme de $\lambda$-proximité existant dans les voisinages à deux sauts d'un nœud. Notre algorithme comporte deux étapes. La première est dédiée au choix du ANS pour atteindre les voisins à un saut. Un nœud appartient à l'ANS du nœud $u$ pour atteindre $v$ si et seulement le lien direct $(u, v)$ ne fait pas partie des meilleurs chemins en terme de probabilité de réception. La deuxième étape permet de sélectionner les nœuds devant appartenir à l'ANS pour atteindre le voisinage à deux sauts. Ces deux étapes se déroulent de manière similaire : le nœud cherche à couvrir chacun de ses voisins ou 2-voisins en utilisant le chemin le plus optimal. Il sélectionne le premier nœud sur ce chemin. S'il existe plusieurs premiers nœuds et qu'aucun n'a déjà été sélectionné, il privilégie celui offrant la meilleure probabilité sur le lien directe. S'il en existe un qui a déjà été sélectionné, il n'en sélectionne pas de supplémentaire.
Lors de la première étape, pour chacun de ses voisins à un saut $v$, le nœud $u$ calcule l'ensemble des meilleurs chemins entre lui et $v$. Il obtient ainsi l'ensemble $f\mathcal{P}_{\mathcal{L}}(u, v)$. Si $v \in f\mathcal{P}_{\mathcal{L}}(u, v)$, cela signifie que le lien direct entre $u$ et $v$ est optimal en terme de $\lambda$-proximité et qu'il n'est pas nécessaire d'ajouter un nœuds à l'ensemble pour atteindre $v$. Sur notre exemple (figure 3.5), $u$ ne sélectionnera donc aucun nœud pour atteindre $v_1$, le lien $(u, v_1)$ étant optimal. Si le lien direct ne donne pas la meilleure probabilité $u$ compare son ANS courant et l'ensemble des chemins optimaux permettant d'atteindre $v$. S'il a déjà sélectionné un nœud $w$ appartenant à l'ensemble des premiers nœuds des chemins optimaux ($w \in f\mathcal{P}_{\mathcal{L}}(u, v) \cap ANS(u)$), il n'est pas nécessaire

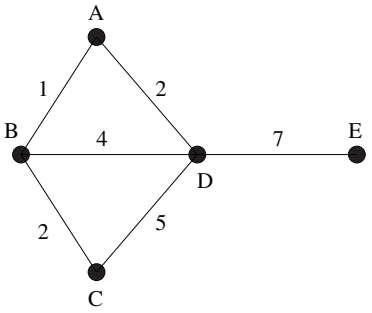



FIGURE 3.6 – *Exemple de mauvais comportement lorsque le dernier lien est un lien limitant concernant la probabilité de transmission correcte. Les nœuds A et B se sélectionnent mutuellement pour couvrir E et créent ainsi une boucle.*

de sélectionner un nouveau nœud. Par exemple, sur la figure 3.5), supposons que $u$ ait d'abord sélectionné $v_1$ pour atteindre $v_4$. Lorsqu'il choisira pour atteindre $v_{10}$ il sélectionnera $v_1$ plutôt que $v_7$ comme il se trouve déjà dans son ANS. Si aucun nœud appartenant à l'ensemble des premiers nœuds des meilleurs chemins n'a été précédemment sélectionnés $f\mathcal{P}_{\mathcal{L}}(u,v) \cap ANS(u) = \emptyset$), $u$ choisit le nœud $w$ tel que $\min_{\prec_L}(f\mathcal{P}_{\mathcal{L}}(u,v)))$. Dans notre exemple, $u$ sélectionnera $v_1$ et non $v_7$ pour couvrir $v_4$ puisque le lien $(u,v_1)$ offre une meilleure probabilité de réception.
La seconde étape de l'algorithme est similaire à la première étape excepté que maintenant $u$ considère son voisinage à deux sauts. Pour chaque voisin à deux sauts $v$, il calcule donc l'ensemble des chemins optimaux menant à $v$ et en déduit l'ensemble $f\mathcal{P}_{\mathcal{L}}(u,v)$. S'il a déjà sélectionné un nœud appartenant également à cet ensemble. Si c'est le cas, il n'est pas nécessaire de sélectionné un nœud supplémentaire. Sur la figure 3.5, aucun autre nœud ne sera sélectionné pour atteindre $v_3$ comme $v_1$ est déjà dans $ANS(u)$ et appartient également à l'ensemble $f\mathcal{P}_{\mathcal{L}}(u,v)$. Si ce n'est pas le cas ($f\mathcal{P}_{\mathcal{L}}(u,v) \cap ANS(u) = \emptyset$), $u$ sélectionne le nœud $w$ dans $f\mathcal{P}_{\mathcal{L}}(u,v)$ tel que le lien direct $(u,w)$ donne la meilleure $\lambda$-proximité. En cas d'égalité, le nœud avec l'identifiant le plus petit est choisi, *i.e.* $u$ sélectionne le nœud $w$ tel que $\max_{\prec_L}(f\mathcal{P}_{\mathcal{L}}(u,v))$. Cependant, cela n'est pas suffisant pour garantir la bonne distribution du message comme illustré sur la figure 3.6. Le nœud $B$ sélectionne $A$ pour atteindre $E$ (le lien $(B,A)$ donne une meilleure probabilité que le lien $(B,C)$ et devra de tout manière être sélectionné pour couvrir $D$). De manière similaire, $A$ sélectionnera $B$ pour atteindre $E$ comme il appartient déjà à son ANS. Cette situation crée un boucle et $E$ devient inatteignable puisque le nœud $D$ est le seul permettant de l'atteindre et qu'il n'a été sélectionné par aucun nœud. Cette situation se produit lorsque le dernier lien est un lien limitant en terme de performances.
Pour pallier ce problème, la condition suivante est ajoutée. Si $u$ possède un identifiant le plus petit sur un chemin optimal, il sélectionne comme ANS un nœud $w$ tel que le chemin $uwv$ existe. Dans notre exemple (figure 3.6), $A$ sélectionne $D$ pour atteindre $E$.

### Évaluation

Nous évaluons les performances de notre algorithme (*FNBP - First Node on Best Path-based QANS selection*) et le comparons à l'approche décrite dans QOLSR (avec l'heuristique MPR-2 [3] adaptée pour conserver la $\lambda$-proximité) et le filtrage de topologie proposé dans [34]. Les nœuds sont déployés dans un carré d'une taille de $1000 \times 1000$ en utilisant processus ponctuel de Poisson avec des densités différentes : le nombre total de nœuds est aléatoire et est obtenu en utilisant une loi de Poisson d'intensité $I$ avec $I = \frac{\delta}{\pi R^2}$. Le rayon de communication $R$ est fixé à 100. Les résultats sont des moyennes sur 100 réseaux dans lesquels une source (notée $u$) et une destination ($v$) sont choisies aléatoirement. Les poids (valeurs de QoS) sur les liens sont tirés aléatoirement

**Input** : $u \in V$, $N(u)$, $N_2(u)$
**Output** : $ANS(u)$
$ANS_1(u) \leftarrow \emptyset$ ;
// Etape 1 : sélection pour atteindre le voisinage à un saut
**foreach** $v \in N(u)$ **do**
  **if** $f\mathcal{P}_{\mathcal{L}}(u,v) \cap ANS_1(u) = \emptyset$ **then**
    **if** $\min_{\prec}(f\mathcal{P}_{\mathcal{L}}(u,v)) \neq v$ **then**
      $ANS_1(u) \leftarrow \min_{\prec_L}(f\mathcal{P}_{\mathcal{L}}(u,v))$ ;
    **end**
  **end**
**end**
$ANS(u) \leftarrow ANS_1(u)$ // Etape 2 : sélection pour atteindre le voisinage à deux sauts
**foreach** $w \in N_2(u)$ **do**
  **if** $f\mathcal{P}_{\mathcal{L}}(u,v) \cap ANS(u) = \emptyset$ **then**
    $ANS(u) \leftarrow \min_{\prec_L}(f\mathcal{P}_{\mathcal{L}}(u,v))$ ;
    **else**
      **if** $\{min_{id}(f\mathcal{P}_{\mathcal{L}}(u,v)) > u\} \wedge \{\exists w \in f\mathcal{P}_{\mathcal{L}}(u,v) \cap N_2(u)\}$ **then**
        $ANS(u) \leftarrow \max_{\prec_L}(f\mathcal{P}_{\mathcal{L}}(u,v) \cap N_2(u))$ ;
      **end**
    **end**
  **end**
**end**

Algorithme 2 – *Sélection des relais par le nœud u.*

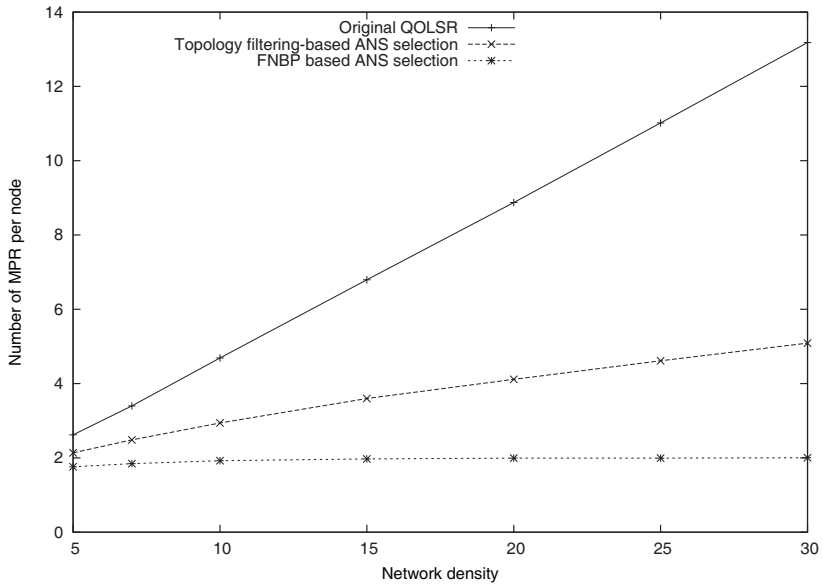



FIGURE 3.7 – *Taille de l'ensemble devant être annoncé dans les messages de contrôle de topologie.*

dans un intervalles fixé. Chaque approche est appliquée sur la même topologie avec la même paire source-destination.
Comme mentionné précédemment, l'objectif de notre algorithme est de fournir des chemins de probabilité de communication maximale tout en minimisant la taille de l'ensemble sélectionné. Nous analysons dans premier temps la taille de l'ensemble sélectionné. Nous calculons ensuite l'overhead engendré par rapport à la solution optimale obtenue en utilisant un algorithme centralisé (algorithme de Dijkstra [12]). Soit $l$ et $l^*$ la somme des logarithmes associée à une route entre $u$ et $v$ obtenue par une des approches évaluée et l'algorithme centralisé respectivement. Nous définissons l'overhead de probabilité comme le ratio $\frac{l-l^*}{l^*}$). Cette overhead correspond en fait à la probabilité "perdue" par rapport à l'optimal.
**Taille de l'ensemble sélectionné** Nous comparons pour chaque approche la taille de l'ensemble sélectionné et devant être annoncé dans les messages de contrôle de topologie. Les résultats sont montrés sur la figure 3.7. Notre approche offre des meilleures performances devant le filtrage de topologie et l'heuristique de QOLSR. Contrairement aux autres approches, la taille de l'ensemble sélectionné reste la même malgré l'augmentation de la densité. Cela est du au fait que lorsque le réseau est dense, nous ne sélectionnons un nouveau nœud si et seulement si il n'existe pas de nœud adéquate déjà sélectionné. Cela n'est pas le cas pour les autres approches qui choisissent automatiquement le premier nœud sur chaque chemin avec une somme de logarithme minimale.
**Overhead** L'overhead des différentes solutions est montré sur la figure 3.8. Notre solution donne les mêmes résultats que le filtrage de topologie, offrant des résultats beaucoup plus intéressant que ceux obtenus avec l'heuristique de QOLSR. L'overhead par rapport à la solution optimale tend à diminuer avec l'augmentation de la densité.

## 3.3 Assignement de portée

### 3.3.1 Définition

Soit $V$ un ensemble de nœuds répartis dans un espace bidimensionnel et $G = (V, E)$ le graphe obtenu lorsque chaque nœud du réseau utilise la portée de communication maximale $R$. $d$ est une fonction de distance de $E$ dans $\mathbb{R}^+$ qui fournit la distance euclidienne entre deux nœuds du

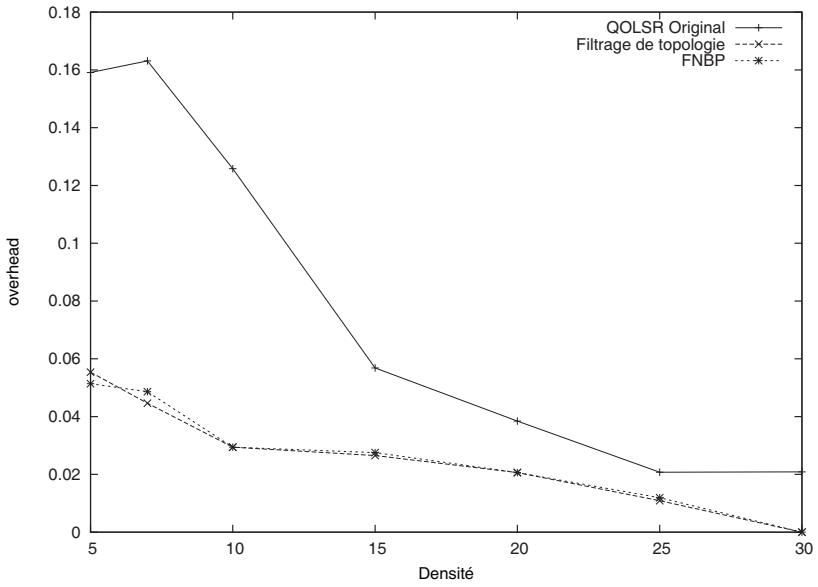



FIGURE 3.8 – *Overhead généré par notre solution (FNBP) comparé à celui généré par les autres solutions. La solution optimale est calculée de manière centralisée.*

réseau. On suppose également que l'on dispose d'une fonction de $\mathbb{R}^+ \times \mathbb{R}^+$ dans $[0;1]$ donnant la probabilité de communication correcte pour une distance donnée.
Un assignement de portée $r$ est une fonction de $V$ dans $\mathbb{R}$ donnant pour chaque nœud la portée de communication qu'il utilise. Le graphe résultant d'un assignement de portée est un graphe orienté noté $G_r = (V, E_r)$. Le coût d'un tel assignement est donné par :

$$C(r) = \sum_{u \in V} c(u)$$

où $c$ est une fonction de $V$ dans $\mathbb{R}$ correspondant au coût pour un nœud d'utiliser une portée particulière. $P_R(s)$ donne la probabilité de communication correcte sur le chemin $s$ lorsque chaque nœud utilise la portée de communication maximale $R$. $P_r(s)$ donne cette même probabilité lorsque chaque nœud du chemin utilise la portée de communication déterminée par l'assignement de portée $r$.
Un assignement de portée est dit *valide* s'il préserve la $\lambda$-proximité *i.e.,* si et seulement si :

$$\begin{aligned} \forall u, v \in V \ (\exists s \in \mathcal{C}_G(u,v) \mid P_R(s) \geq \lambda) \\ \Rightarrow (\exists s' \in \mathcal{C}_{G_r}(u,v) \mid P_r(s') \geq \lambda)). \end{aligned}$$

**Definition 3.2.** *Ce problème est NP-dur pour un coût fonction de la portée et une fonction de probabilité quelconque.*

En effet, ce problème peut être réduit à celui du problème d'assignement de portée bidimensionnel étudié dans [30]. Ce dernier consiste à trouver une portée de communication à chaque nœud d'un réseau, répartis dans un espace 2D, qui minimisent l'énergie totale consommée avec pour contrainte que le graphe obtenu en utilisant le modèle UDG doit être connexe.
Soit $p$ une fonction de probabilité telle que :

$$P_r(x) = \begin{cases} 1 & \text{si} \quad x < r, \\ 0 & \text{sinon.} \end{cases}$$

Si l'on utilise cette fonction de probabilité dans notre problème, on aboutit au problème décrit dans [30]. Dans [11], les auteurs montrent que ce problème est NP-dur. Étant donné que notre problème peut se réduire à celui de l'assignement de portée bidimensionnel, il est également NP-dur.

### 3.3.2 Évaluation d'un assignement

Nous définissons plusieurs fonctions de coût permettant d'évaluer la qualité d'un assignement de portée. La première est celle utilisée classiquement dans la littérature. Elle se base uniquement sur le coût de la portée choisie par le nœud et s'appuie sur la modélisation du coût énergétique d'envoyer un paquet introduite dans [19]. Le coût $E(u)$ d'envoi d'un paquet par le nœud $u$ utilisant la portée $r(u)$ est défini comme suit :

$$E(u) = r(u)^{\alpha} + c,$$

où $c$ une constant représentant le coût fixe (traitement de l'envoi). Minimiser le coût énergétique d'un assignement de portée dans un réseau revient alors à minimiser la somme des portées utilisées :

$$\min\left(\sum_{u\in V} E(u)\right) \Leftrightarrow \min\left(\sum_{u\in V} r(u)^{\alpha}\right).$$

Le coût $c(u)$ est dans ce cas défini par :

$$c(u) = r(u)^{\alpha}.$$

L'inconvénient de cette fonction est qu'elle ne prend pas en compte le trafic supporté par le nœud. Un nœud supportant beaucoup de trafic enverra plus de paquets et devra traiter plus de paquets. Généralement, on peut repérer ces nœuds par leur degré. Un degré élevé suppose en effet que tous les messages envoyés par les nœuds environnant devront être traités, ne serait ce que pour constater qu'ils ne sont pas destinés à ce nœud. De plus un nœud avec un degré élevé est plus sujet aux interférences (la réciproque n'étant pas exacte). L'idée des fonctions de coût qui suivent est de prendre en compte ce fait en affectant un coût plus élevé pour l'utilisation d'une certaine portée par un tel nœud qu'il ne le serait pour un nœud n'ayant peu de paquets à router. La fonction basée sur le degré part de ce postulat et encourage les nœuds avec un degré élevé à utiliser une portée de communication faible :

$$c(u) = \delta(u) r(u)^{\alpha}.$$

Cette fonction constitue une approximation du trafic supporté par un nœud.

Nous proposons une autre approximation de ce trafic utilisable lorsque le trafic se fait des exclusivement des nœuds vers un puits unique. L'idée de cette fonction est de prendre en compte la distance du nœud au puits et le trafic qu'il peut potentiellement router vers ce dernier pour les nœuds se trouvant plus éloigné du puits.
Considérons le nœud $u$ qui se trouve à $h$ sauts du puits. On définit $N^{+}(u)$ comme étant l'ensemble des voisins de $u$ se trouvant à distance $h+1$ du puits. De façon similaire, $N^{-}(u)$ est l'ensemble de ses voisins se trouvant à $h-1$ sauts du puits. Le coût d'un assignement de portée au niveau du nœud $u$ est donné par :

$$c(u) = \rho(u) r(u)^{\alpha},$$

où $\rho$ est une fonction récursive de $V$ dans $\mathbb{R}$ représentant le trafic supporté par un nœud :

$$\rho(u) = \begin{cases} 1 & \text{si } N^{+}(u) = \emptyset, \\ \sum_{v\in N^{+}(u)} \frac{\rho(v)}{|N^{-}(v)|} p_{r(v)}(d(u,v)) & \text{sinon.} \end{cases}$$

Si un nœud n'a pas de voisins à $h+1$ sauts ($N^+(u) = \emptyset$) le seul trafic qu'il supporte est celui qu'il génère. Dans le cas contraire, il faut prendre en compte le trafic généré par chacun de ses voisins à $h+1$. Pour chaque voisin $v$ de $N^+(u)$, nous définissons le trafic pris en charge par $u$ pour $v$ comment étant une partie du trafic géré par $v$. Cette partie est dépendante du nombre de voisins de $v$ qui peuvent potentiellement router ses paquets et de la probabilité qu'à $u$ de recevoir les paquets envoyés par $v$.

## 3.4 Approches centralisées

La première approche centralisée que nous décrivons est une approche gloutonne qui assigne à chaque nœud la portée minimale permettant de préserver la $\lambda$-proximité. Après avoir analyser son comportement, nous montrons comment il est possible de l'améliorer.

### 3.4.1 Approche gloutonne

#### Description

On note $\overline{P_R}(i,j)$ la probabilité de communication correcte du meilleur chemin entre $i$ et $j$ en utilisant la portée maximale de communication ($R$). Soit $I_u$ l'ensemble des chemins dans lesquels le nœud $u$ est impliqué comme source ou relais et dont la probabilité de communication correcte est supérieure à $\lambda$ :

$$I_u = \{s_{i,j} \mid u \in s_{i,j}\ j \wedge (P_R(s_{i,j}) = \overline{P_R}(i,j)) \wedge (P(s_{i,j} \geq \lambda)\}.$$

Lors de l'assignement d'un portée au nœud $u$, la contrainte à considérer pour chaque chemin $s = a_1, \ldots, a_k, u, b_1, \ldots, b_l \in I_u$ dans lequel $u$ est impliqué est la suivante :

$$\begin{aligned} \forall s \in I_u \qquad & \prod_{i=1}^{k} p_{r(a_i)}(d(a_i, a_{i+1})) \times p_{r(u)}(d(u,b_1)) \times \prod_{i=1}^{l-1} p_{r(b_i)}(d(b_i, b_{i+1})) \geq \lambda \\ \Leftrightarrow \quad & \forall s \in I_u \left( \frac{P_R(s)}{p_R(d(u,b_1))} \times p_{r(u)}(d(u,b_1)) \geq \lambda \right). \end{aligned}$$

Pour préserver la $\lambda$-proximité d'un chemin $s$, la portée de communication $r(u)$ devant être assignée à $u$ doit respecter :

$$p_{r(u)}(d(u,b_1)) \geq \frac{\lambda}{P_R(s)} \times p_R(d(u,b_1)).$$

La portée correspondante peut être trouvée en utilisant la fonction inverse de LNS :

$$r(u) = r(d(u,b_1), \frac{\lambda}{P_R(s)} \times p_R(d(u,b_1))).$$

L'algorithme centralisé (algorithme 3 se contente d'assigner à chaque nœud la portée permettant de garantir la $\lambda$-proximité de chacun des chemins dans lesquels il est impliqué. A chaque fois que la portée d'un nœud est modifiée, l'ensemble de ces chemins est mis à jour avec cette nouvelle portée. Un des choix pouvant avoir un impact sur les performances de cet algorithme est le tri de l'ensemble $V$. Nous étudions deux approches : trier l'ensemble des nœuds dans l'ordre croissant de leur implication ($|I_u|$) ou dans l'ordre décroissant. Intuitivement, l'avantage d'utiliser l'ordre décroissant est qu'une portée de communication relativement faible sera assignée aux nœuds subissant le plus de trafic.

**Lemme 3.1.** *Les deux approches de l'algorithme glouton préservent la $\lambda$-proximité.*

**Input** : $V$, $R$ rayon de communication maximal
**Output** : $r : V \mapsto \mathbb{R}$
trier $V$ en fonction $|I_u|$ (par ordre croissant ou décroissant) ;
**foreach** $u \in V$ **do**
  $maxR = 0$ ;
  **foreach** $s = a_1, \ldots, a_k, u, b_1, \ldots, b_l \in I_u$ **do**
    $r =$ portée minimale telle que : $p_r(d(u, b_1)) = \frac{\lambda}{P_R(s)} \times p_R(d(u, b_1))$ ;
    **if** $r > maxR$ **then**
      $maxR = r$ ;
    **end**
  **end**
  $r(u) = maxR$ ;
  mettre à jour les chemins de $I_u$ avec la nouvelle portée de $u$ ;
**end**

Algorithme 3 – *Centralized Greedy Algorithm for Range Assignment*

*Démonstration.* Considérons un chemin $s = a_1, \ldots, a_k, u, b_1, \ldots, b_l$ tel que $P_R(s) \geq \lambda$. Si $P_R(s) = \lambda$, alors aucun nœud impliqué dans $s$ ne modifiera sa portée de communication. Si $P_R(s) > \lambda$, on note $u$ le premier nœud modifiant sa portée. La nouvelle probabilité associée au chemin $s$ sera :

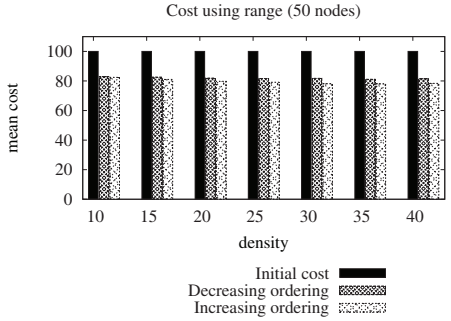
$$\begin{aligned} P_r(s) &= \prod_{i=1}^{k-1}(p_R(d(a_i, a_{i+1}))) \times p_R(d(a_k, u)) \\ &\quad \times p_{r(u)}(d(u, b_1)) \times \prod_{i=1}^{l-1}(p_R(d(b_i, b_{i+1}))) \\ &= \frac{P_R(s)}{p_R(d(u, b_1))} \times P_{r(u)}(d(u, b_1)) \\ &= \frac{P_R(s)}{p_R(d(u, b_1))} \times \frac{\lambda}{P_R(s)} \times p_R(d(u, b_1)) \\ &= \lambda. \end{aligned}$$

Comme $P_r(s) = \lambda$, aucun autre nœud ne modifie sa portée de transmission et l'ajustement préserve la $\lambda$-proximité. □

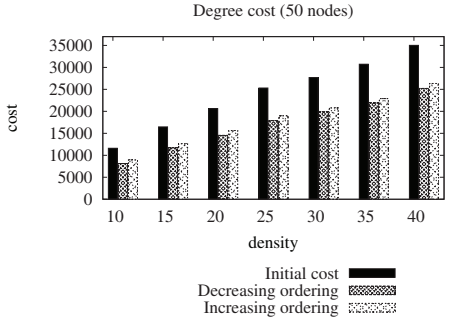
### Évaluation

L'évaluation des différentes approches que nous proposons se fait à l'aide du logiciel Scilab. Le scénario considéré est celui où des nœuds statiques sont déployés sur une surface plane (un carré en l'occurrence) dont la taille dépend de la densité désirée. La portée de communication maximale $R$ est de 1. Nous faisons varier la densité en changeant la surface de déploiement mais le nombre de nœuds est toujours de 70. Nous prenons $\lambda = 0.25$. Les deux variantes de l'algorithme glouton (tri par ordre croissant et décroissant) sont appliquées sur chaque réseau généré. Pour chaque assignement de portée obtenu nous calculons les différents coûts définis précédemment.

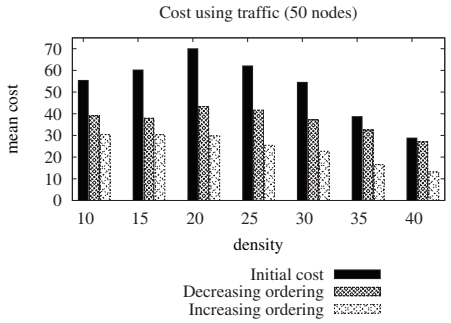
**Coût utilisant la portée** Le coût utilisant la fonction dépendant uniquement de la portée de communication (Figure 3.9(a)) choisie est pratiquement le même lors de l'utilisation des tris croissant et décroissant. En considérant cette fonction de coût, l'assignement de portée permet une économie d'énergie de 30%.




(a) $\sum r(u)^\alpha$




(b) $\sum \delta(u) r(u)^\alpha$




(c) $\sum \rho(u) r(u)^\alpha$

FIGURE 3.9 – *Résultats obtenus en utilisant l'approche gloutonne.*

**Coût utilisant le degré** Une fois encore, les deux variantes ont des performances similaires(Figure 3.9(b)). Cependant, cette fois-ci le tri décroissant a des résultats légèrement supérieur au tri croissant. Cela peut s'expliquer par le fait que les nœuds avec un degré élevé ont un ensemble d'implication $I_u$ de cardinalité plus importante, et seront donc traités en premier.

**Coût utilisant le trafic** On peut constater des performances assez différentes entre les deux approches (Figure 3.9(c)). Le tri par ordre décroissant obtient de meilleures performances quelque soit la densité du réseau. De manière surprenante, le coût a tendance a baissé lorsque la densité atteint 35 nœuds par zone de communication. Cela peut s'expliquer par la méthode utilisée pour générer le réseau. Avec une densité de 30 ou plus, la distance en nombre de sauts entre le puits et tout autre nœud est au maximum de 2.

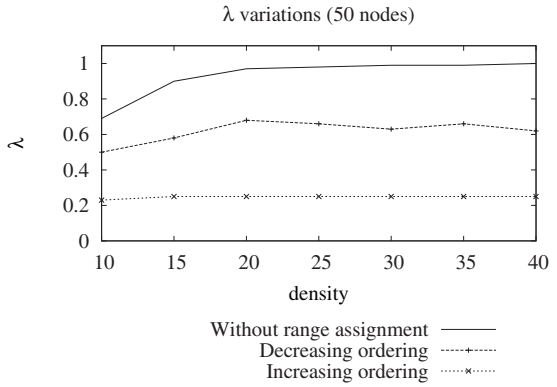
Dans l'optique d'avoir une idée plus précise du comportant de l'approche gloutonne, la figure 3.10 présente des résultats supplémentaires sur l'assignement de portée.

**$\lambda$-proximité après l'ajustement de portée** La valeur moyenne de $\lambda$ tend vers 1 lorsque la densité augmente (Figure 3.10(a)). Le tri croissant est le seul à approcher la densité cible de 0.25. Cela signifie que le tri décroissant ne profite pas totalement de la possibilité qui lui est offerte de diminuer la $\lambda$-proximité.

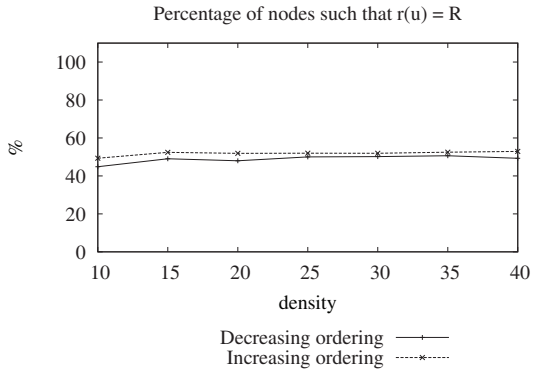
**Pourcentage de nœuds conservant la portée maximale** Le pourcentage de nœuds ayant toujours la portée de communication optimale est assez élevé, de l'ordre de 50% (Figure 3.10(b)). Cela peut s'expliquer par le fait que lorsque l'on considère un nœud et qu'on ajuste sa portée par rapport à un chemin critique, tous les nœuds sur ce chemin ne pourront pas modifier la leur.

### 3.4.2 Versions améliorées

Le principal inconvénient des versions initiales est qu'une fois qu'une portée a été assignée à un nœud, tout les nœuds de son chemin critique ne peuvent modifier la leur. Nous proposons




(a) Valeur moyenne de $\lambda$ avant et après l'assignement de portée (valeur cible $\lambda = 0.25$).




(b) Percentage of nodes which keeps the initial transmission power $R$.

FIGURE 3.10 – *$\lambda$-proximité*

deux nouvelles approches prenant en compte cet inconvénient. Dans celles-ci un nœud choisit une portée de transmission qui n'empêche pas les autres nœuds de modifier la leur.

### Algorithme équitable

Dans cette version, la première étape reste la même : l'ensemble d'implication $I_u$ de chaque nœud $u$ est calculé. La nouvelle portée de transmission d'un nœud dépend toujours de son chemin critique mais lors du choix d'une nouvelle portée le fait que d'autres noeuds du chemin voudront éventuellement ajuster leur portée par rapport à ce chemin est pris en compte. L'objectif est d'obtenir un chemin dont la probabilité de communication correcte sera de $\lambda$. Chaque nœud peut donc profiter d'une part de la « marge » disponible pour atteindre $\lambda$. Soit $M$ cette marge disponible pour un chemin $s$ est calculée en fonction de la probabilité obtenue lors de l'utilisation de la portée maximale par tout nœud :

$$M = \frac{\lambda}{P_R(s)}.$$

La probabilité de communication correcte sur chaque lien de $s$ peut alors être dimininuée de $\sqrt[|s|]{M}$, $|s|$ étant le nombre de liens de $s$. L'algorithme 4 est l'algorithme équitable. On peut remarquer qu'il n'est pas nécessaire de mettre à jour l'ensemble d'implication des nœuds étant donné que la marge disponible pour un nœud d'un chemin est calculée a priori.

### Version basée sur le degré

La version de l'algorithme basée sur le degré (algorithme 5 emploie la même technique que la version équitable pour s'assurer que chaque nœud profite de l'assignement de portée. Dans cette variante, l'intensité de la baisse de la portée est dépendante du degré du nœud. L'idée est d'avantager les nœuds ayant un degré relativement élevé comme ils seront probablement plus sollicités lors des opérations de routage. Plus le degré d'un nœud est important, plus il profitera de l'assignement de portée. La probabilité de communication correcte sur chaque lien du chemin $s$ peut être diminuée de $\sqrt[\Delta(s)]{M}^{\delta(u)}$, $\delta(u)$ étant le degré du nœud émetteur et $\Delta$ la somme des degrés des nœuds du chemin $s$ (excepté le destinataire, dont la puissance d'émission n'entre pas en jeu).

**Lemme 3.2.** *L'approche équitable et l'approche basée sur le degré maintiennent la $\lambda$-proximité.*

**Input** : $V$, $R$ portée de communication maximale
**Output** : $r : V \mapsto \mathbb{R}$
trier $V$ selon $|I_u|$ (ordre croissant ou décroissant) ;
**foreach** $u \in V$ **do**
  $maxR = 0$ ;
  **foreach** $s = a_1, \ldots, a_k, u, b_1, \ldots, b_l \in I_u$ **do**
    $r =$ portée de communication telle que : $p_r(d(u, b_1)) = \frac{P_R(s)}{\sqrt[|s|]{\frac{P_R(s)}{\lambda}}}$ ;
    **if** $r > maxR$ **then**
      $maxR = r$ ;
    **end**
  **end**
  $r(u) = maxR$ ;
**end**

Algorithme 4 – *Algorithme d'assignement de portée basé sur l'équité.*

**Input** : $V$, $R$ portée de communication maximale
**Output** : $r : V \mapsto \mathbb{R}$
trier $V$ selon $|I_u|$ (ordre croissant ou décroissant) ;
**foreach** $u \in V$ **do**
  $maxR = 0$ ;
  **foreach** $s = a_1, \ldots, a_k, u, b_1, \ldots, b_l \in I_u$ **do**
    $r =$ portée de communication telle que : $p_r(d(u, b_1)) = \frac{P_R(s)}{\left(\sqrt[\Delta(s)]{\frac{P_R(s)}{\lambda}}\right)^{\delta(u)}}$ ;
    **if** $r > maxR$ **then**
      $maxR = r$ ;
    **end**
  **end**
  $r(u) = maxR$ ;
**end**

Algorithme 5 – *Algorithme d'assignement de portée basé sur le degré.*

*Démonstration.* Nous montrons que l'approche équitable est correcte, le raisonnement étant le même pour l'approche basée sur le degré. Soit un chemin $s = a_1, \ldots, a_{k+1}$ tel que $P_R(s) \geq \lambda$. Si tous les nœuds ajustent leur puissance de transmission en utilisant $s$ comme chemin critique, la nouvelle probabilité associée à $s$ sera :

$$\begin{aligned} P_r(s) &= \prod_{i=1}^{k} p_{r(a_i)}(d(a_i, a_{i+1})), \\ &= \prod_{i=1}^{k} \frac{P_R(s)}{\sqrt[|s|]{\frac{P_R(s)}{\lambda}}} = \left( \frac{P_R(s)}{\sqrt[|s|]{\frac{P_R(s)}{\lambda}}} \right)^{|s|}, \\ &= \lambda. \end{aligned}$$

Il en résulte que la $\lambda$-proximité est conservée. □

L'évaluation de ces solutions est faite sur les mêmes réseaux que ceux utilisés pour l'évaluation de la version gloutonne.

**Coût utilisant la portée** En utilisant la portée comme métrique, les deux versions ont des meilleures performances que l'approche gloutonne (Figure 3.11(a)). On peut constater une baisse de 40% du coût avec une densité de 10. Cette baisse peut aller jusqu'à 65% avec une densité de 40.

**Coût utilisant le degré** Les performances des versions améliorées sont encore une fois meilleures que celles de la version gloutonne (Figure 3.11(b)). Le coût utilisant le degré comme indicateur diminue d'au moins 50% avec une densité de 10.

**Coût utilisant le trafic** Le coût utilisant le trafic comme métrique montre de meilleures performances qu'avec la version gloutonne. On constate qu'il est d'au plus 20% du coût obtenu sans assignement de portée.

**$\lambda$-proximité après l'ajustement de portée** La $\lambda$-proximité obtenue après l'ajustement de portée est montrée sur la figure 3.12(a). Elle est toujours plus élevée que la $\lambda$-proximité ciblée (0.25). Il serait donc toujours possible de diminuer la portée de certains nœuds.

**Pourcentage de nœuds conservant la portée maximale** Avec ces deux approches, tous les nœuds benificient de l'assignement et dimininuent leur portée.

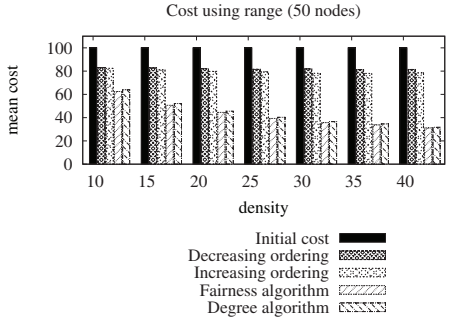
Ces algorithmes sont encore améliorables. Cependant, de par leur nature centralisée, ils ne sont pas utilisables dans un réseau de capteurs. L'idéal serait de trouver une version localisée de ces algorithmes. Cependant, il n'existe aucune solution localisée à ce problème.

**Lemme 3.3.** *L'assignement de portée préservant la $\lambda$-proximité n'a pas de solution localisée.*

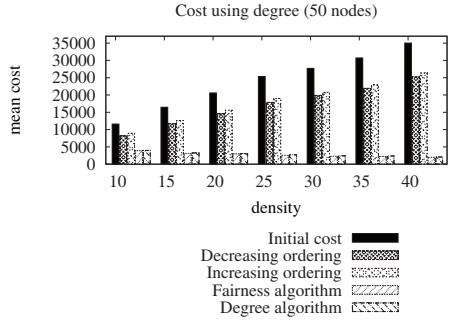
*Démonstration.* Supposons qu'un algorithme localisé (utilisant des informations à $k$ sauts) préservant la $\lambda$-proximité existe et qu'il soit appliqué par le nœud $u$ du réseau $G = (V, E)$. Il est toujours possible de construire un réseau $G' = (V \cup \{w\}, E')$ avec $w$ un nœud à $k+1$ sauts de $u$ et tel que $P_R(u, w) = \lambda$ et $P_{r(u)}(u, v) < \lambda$ si $r(u) \neq R$. La solution donnée par l'algorithme pour le graphe $G'$ sera la même que celle donnée pour $G$. Donc, à moins que $\forall u, r(u) = R$ l'assignement de portée donné par l'algorithme pour le $G'$ n'est pas valide. Il en résulte qu'aucun algorithme localisé n'existe pour ce problème. □

## 3.5 Problème localisé

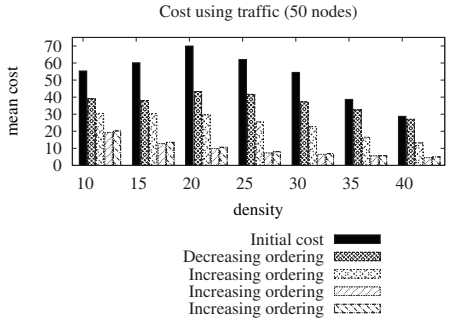
Nous introduisons la notion de $\lambda$-proximité localisée. Cette définition s'applique sur l'ensemble du réseau, mais un nœud peut s'assurer qu'il préserve la $\lambda$-proximité localisée dans son voisinage à $k$ sauts. L'idée est de prendre en compte la distance en nombre de sauts entre deux nœuds pour déterminer s'ils sont $\lambda$-proche.




(a) $\sum r(u)^{\alpha}$

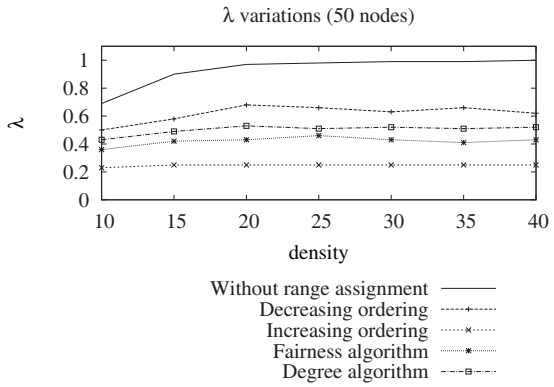



(b) $\sum \delta(u) r(u)^{\alpha}$

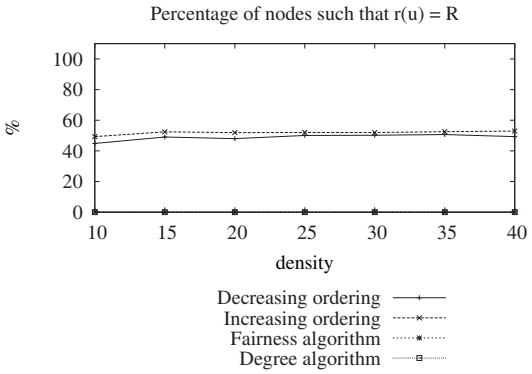



(c) $\sum \rho(u) r(u)^{\alpha}$

FIGURE 3.11 – *Résultats obtenus en utilisant les versions gloutonnes et améliorées. La version améliorée donne des meilleures résultats que la version gloutonne.*




(a) Valeur moyenne de $\lambda$ avant et après assignement de portée (valeur cible $\lambda = 0.25$).




(b) Pourcentage de nœuds conservant la puissance de transmission initiale $R$.

FIGURE 3.12 – *Étude de la $\lambda$-proximité avant et après l'ajustement de portée pour les différentes approches.*

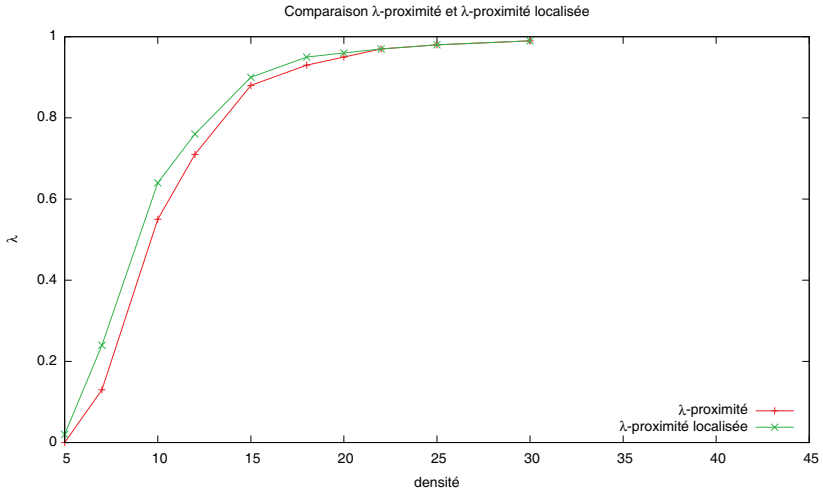



FIGURE 3.13 – *Comparaison entre la λ-proximité et la λ-proximité moyenne.*

**Definition 3.3.** *Un réseau est dit* λ-proche localement *si et seulement si il existe entre chaque nœud du réseau un chemin dont la probabilité de communication correcte est d'au moins* $\lambda^{h(u,v)}$, $h(u,v)$ *étant la distance en nombre de sauts entre* $u$ *et* $v$.

Un réseau est donc dit *λ-proche localement* si et seulement si :

$$\forall u, v \in V \ \exists s \in \mathcal{C}_G(u,v) \mid P(s) \geq \lambda^{h(u,v)}.$$

**Input** : $G = (V, E)$
**Output** : $\lambda$
$\lambda = 1$ ;
**foreach** $u \in V$ **do**
  $h =$ distance minimale en nombre de sauts entre $u$ et $v$ ;
  $p = \max_{s \in \mathcal{C}_G(u,v)} P(s)$ ;
  $\lambda_{u,v} = \sqrt[h]{p}$ ;
  $\lambda = \min(\lambda_{u,v}, \lambda)$ ;
**end**

Algorithme 6 – *Calcul de la λ-proximité localisée.*

La figure 3.13 compare la λ-proximité localisée (algorithme 8) et la λ-proximité obtenues en utilisant le modèle LNS. On constate que les résultats sont relativement proches et qu'une λ-proximité localisée élevée dénote une λ-proximité élevé.
Dans cet optique, un assignement de portée est dit *valide* s'il préserve la λ-proximité localisée :

$$\forall u, v \in V \ (\exists s \in \mathcal{C}_G(u,v) \mid P_R(s) \geq \lambda^{h(u,v)}) \\ \Rightarrow (\exists s' \in \mathcal{C}_{G_r}(u,v) \mid P_r(s') \geq \lambda^{h(u,v)})).$$

## 3.6 Solutions localisées

Avec une telle définition, il est possible de concevoir des algorithmes préservant la λ-proximité localisée. Nous présentons deux approches. La première est l'approche naïve : chaque nœud ajuste

sa portée telle que la probabilité de communication avec chacun de ses voisins soit supérieure ou égale à $\lambda$. La seconde se base sur la réduction de graphe en utilisant le RNG.

### 3.6.1 Description

#### Algorithme glouton

L'algorithme glouton se base sur un principe simple : si le réseau était $\lambda$-proche localement, alors en ajustant sa portée de communication telle que la probabilité de communication soit d'au moins $\lambda$, il le restera ensuite.

**Input** : $u, N(u)$
**Output** : $r(u)$ : portée de communication utilisée par $u$.
$p_{min} = 1$ ;
$v_{min} = -1$ ;
**foreach** $v \in N(u)$ **do**
  **if** $(p(u,v) \geq \lambda) \wedge (p(u,v) \leq p_{min}))$ **then**
    $p_{min} = p(u,v)$ ;
    $v_{min} = v$ ;
  **end**
**end**
$r(u)$ = portée de communication telle que $p_{r(u)}(u, v_{min}) = \lambda$ ;

Algorithme 7 – *Calcul de la $\lambda$-proximité localisée.*

#### Approche basée sur une réduction de graphe

La deuxième approche utilise l'algorithme de réduction de graphe RNG présentée précédemment. Chaque nœud calcule son graphe de voisinage relatif. Si aucun nœud ne permet d'éliminer un lien avec un voisin direct, la portée de communication est ajustée de manière à atteindre ce nœud avec une probabilité $\lambda$. Si un lien $(u, v)$ peut être supprimé grâce à un nœud $w$, le nœud adapte sa portée de telle sorte que la probabilité de communication correcte avec $w$ soit d'au moins $\sqrt{\lambda}$. Le nœud indiquera ensuite à $w$ qu'il lui faut maintenir une probabilité $p(w, v)$ d'au moins $\sqrt{\lambda}$. La probabilité de communication correcte du chemin $s = u, w, v$ sera alors de $\lambda$.

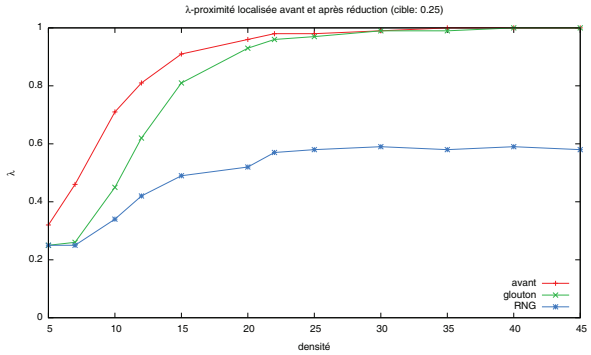
### 3.6.2 Évaluation

L'évaluation s'effectue de la même manière que pour les versions centralisées. Les résultats sont montrés sur la figure 3.14. La $\lambda$-proximité localisée ciblée est de 0.25. On constate que l'approche gloutonne donne de moins bons résultats que l'approche basée sur le RNG.

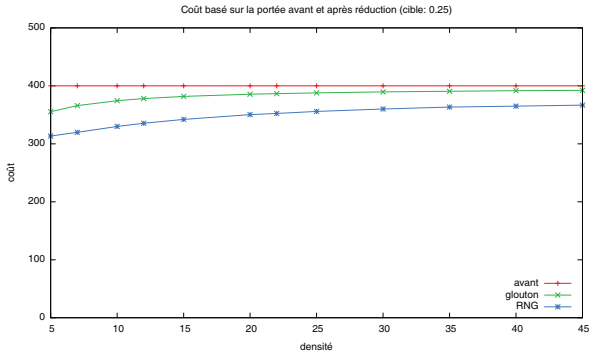
On peut constater que la $\lambda$-proximité localisée obtenue avec la version gloutonne rejoint assez rapidement celle du graphe original. Cela s'explique par le fait que plus la densité est élevée, plus les chances d'avoir un nœud en bordure sont importantes, limitant ainsi de réduire la portée puisque tous les voisins sont conservés. L'approche basée sur le RNG offre de meilleurs résultats et la $\lambda$-proximité localisée finit par se stabiliser. Cela est dû au fait que les liens les moins performants sont enlevés et la portée peut être adaptée au nœud sélectionné comme voisins RNG.

**Input** : $u, N(u), N_2(u)$
**Output** : $r(u)$ : portée de communication utilisée par $u$.
$r(u) = 0$ ;
$S = \emptyset$ ;
**foreach** $v \in N(u)$ **do**
  **if** $\exists w \in N(u) \mid (P(u,w) \times P(w,v)) > P(u,v)$ **then**
    $w_{max} = \max(p(u,w), w \in N(u) \mid (P(u,w) \times P(w,v)) > P(u,v))$ ;
    $r_w =$ portée de communication telle que $p_{r(u)}(u, w_{max}) = \sqrt{\lambda}$ ;
    $r(u) = \max(r(u), r_w)$ ;
    $S = S \cup \{w_{max}\}$ ;
  **end**
  **else**
    $r_v =$ portée de communication telle que $p_{r(u)}(u,v) = \lambda$ ;
    $r(u) = \max(r(u), r_v)$ ;
  **end**
**end**
$r(u) =$ portée de communication telle que $p_{r(u)}(u, v_{min}) = \lambda$ ;

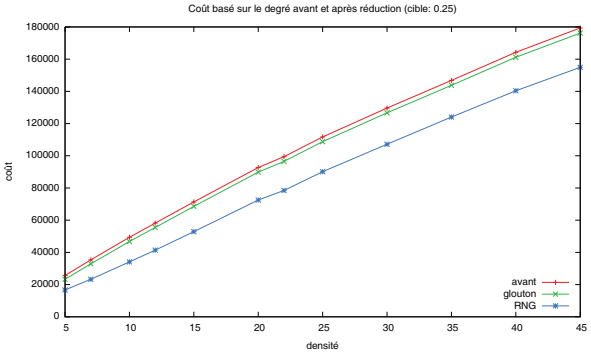
Algorithme 8 – *Première phase de l'approche basée sur la réduction de graphe.*




(a) $\lambda$-proximité localisée




(b) coût basé sur la portée




(c) coût basé sur le degré

FIGURE 3.14 – *$\lambda$-proximité localisée, coût basé sur la portée et coût basé sur le degré avant et après assignement de portée. Les résultats sont obtenus sur 1000 graphes de 100 nœuds dont la position est générée aléatoirement.*

## 3.7 Conclusion

Dans ce chapitre nous avons introduit la notion de $\lambda$-proximité pour juger de la qualité d'une topologie lors de l'utilisation d'une couche physique réaliste. Notre démarche a ensuite été de proposer une méthode d'adaptation des algorithmes de réduction de graphe dans le cadre de l'utilisation d'un modèle probabiliste, démarche que nous avons illustrée avec le graphe de voisinage relatif.
Nous avons ensuite considéré le problème d'assignement de portée lors de l'utilisation d'une couche physique probabiliste. Après avoir montré que ce problème est NP-dur et qu'il ne possède pas de solution localisée, nous avons introduit la notion $\lambda$-proximité localisée. Celle-ci est en concordance avec la $\lambda$-proximité : plus la $\lambda$-proximité localisée est élevée, plus la $\lambda$-proximité l'est aussi. Nous avons étudié deux approches localisées au problème du maintien de la $\lambda$-proximité localisée lors d'un assignement de portée. La première, gloutonne, présente l'avantage d'être simple et de ne nécessiter qu'une connaissance du voisinage à un saut. La deuxième approche proposée se base sur le graphe de voisinage relatif et offre de meilleures performances, mais nécessite une connaissance du voisinage à deux sauts si la probabilité est indépendante de la distance. De plus, elle nécessite l'envoi d'un message par nœud une fois sa décision de portée effectuée. D'autres améliorations pourraient être apportées à cette dernière approche, notamment dans le choix de l'élimination ou non d'un lien.

# 4 Exemple de pile de communication : Goliath

Sommaire



Dans ce chapitre, nous présentons Goliath (Generic and Optimized LIghtweight stack for Ambiant TecHnologies), la pile de communication que nous avons implémentée de manière à tester les différents protocoles dont nous avons parlé dans les chapitres précédents. Elle s'appuie sur le mini noyau temps réel *FreeRTOS*[1] (Free Real Time Operating System). La première section fournit une vue générale de Goliath. Nous donnons ensuite les éléments technologiques spécifiques mis en place dans Goliath mais ne faisant pas partie de la pile protocolaire proprement dit. Nous décrivons ensuite la couche physique et la couche MAC utilisées. Les sections suivantes décrivent respectivement la découverte de voisinage de Goliath, le protocole de routage et les protocoles de transport. Enfin, nous présentons une évaluation de Goliath avant de conclure.

1. http ://www.freertos.org/

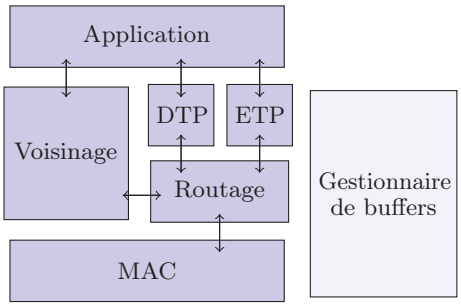



FIGURE 4.1 – *Architecture générale de Goliath.*

## 4.1 Vue générale

L'architecture générale de GOLIATH est montrée dans la figure 4.1. La pile protocolaire est découpée en plusieurs blocs, chacun ayant un rôle spécifique. Une application désirant envoyé un message à un autre nœud le passe à la couche transport en indiquant quel protocole elle souhaite utiliser pour l'envoi de ce message. GOLIATH propose deux protocoles de transport ayant des niveaux de fiabilité différents. Le plus basique est DTP (*Datagram Transport Protocol*). Pour les messages les plus importants, l'application peut choisir d'utiliser ETP (*End-to-end Transport Protocol*) qui fournit un accusé de réception de bout en bout. La couche transport traite le message et passe le paquet à envoyer à la couche routage qui s'occupe de trouver un chemin vers la destination du paquet. Une fois qu'elle a rempli les en-têtes qu'elle utilise, elle envoie le paquet à la couche MAC. Cette dernière est responsable de l'accès au medium, c'est-à-dire de désigner quel nœud peut y accéder à un instant particulier.

Dans l'esprit du *zero copy*, un gestionnaire de buffers permet aux différentes couches de la pile d'éviter de recopier les données qu'elles doivent traiter dans des buffers qui leur seraient propres. Comme mentionner précédemment, nous utilisons FreeRTOS pour l'implémentation de GOLIATH. Une description détaillée de FreeRTOS est disponible dans le chapitre 2, section 2.1.4. Concernant la gestion mémoire de, étant donné que nous ne créons pas et ne supprimons pas de tâches dynamiquement, nous choisissons d'utiliser le mécanisme de gestion de mémoire le plus léger ne permettant pas de libérer la mémoire précédemment allouée.
Concernant la représentation du temps, nous nous affranchissons du timer de FreeRTOS qui boucle trop rapidement. Nous choisissons d'utiliser pour les timers propres à GOLIATH une autre variable *time* dont une unité représente une demi-seconde.

## 4.2 Éléments technologiques spécifiques

Dans cette section nous détaillons les mécanismes adjacents à la pile protocolaire que nous avons mis en place dans GOLIATH mais ne faisant pas partie de la pile protocolaire proprement dit.

### 4.2.1 Support de communication asynchrone

#### Motivations

Lorsqu'une tâche désire passer un paquet à une autre couche pour que celle-ci le traite, elle doit lui indiquer plusieurs éléments. Prenons l'exemple de la communication entre la couche MAC et la couche routage en supposant que la couche MAC désire indiquer à la couche routage qu'elle vient de recevoir un paquet à traiter. Les éléments à transmettre dans ce cas sont la taille des données utiles à la couche routage et la source du paquet (l'identifiant de l'expéditeur). Un mécanisme

d'appels de fonction serait préjudiciable. En effet, il ne permettrait pas de distinguer le travail effectué par chaque tâche. Dans l'exemple que nous avons pris, la couche MAC appellerait une fonction de la couche routage : c'est donc la tâche de la couche MAC qui exécuterait alors des fonctions de routage.

#### Mise en œuvre

Nous proposons d'utiliser, pour la communication entre couches le mécanisme de files de FreeRTOS. Chaque protocole possède une file dans laquelle les autres protocoles mettent des données indiquant les actions attendues. Toutes les files que nous utilisons contiennent des messages dont la taille est limitée à un octet. Ce choix est justifié par le fait que l'utilisation d'une file entraîne deux copies des objets qu'elle peut contenir : une copie lorsque l'objet est empilé dans la file, et une copie lorsqu'il est dépilé. Cette file est également utilisée pour les commandes internes à la couche comme la programmation de tâches à effectuer à une date ultérieure. Chaque couche reconnaît certains types de message asynchrone. Nous adoptons la notation suivante pour le découpage de cet octet :

$$(\text{description}_1 : n_1) \dots (\text{description} : n_k),$$

avec $\text{description}_1$ la description de ce à quoi sont affectés les $n_1$ premiers bits. Une commande prenant en paramètres 6 bits s'écrit alors :

$$(\text{commande} : 2)\ (\text{paramètres} : 6)$$

La commande est constituée de 2 bits et indique l'action à effectuer. Il peut s'agir par exemple du traitement d'un paquet venant de la couche supérieure, du traitement d'un paquet venant d'une couche inférieure. Les éventuels paramètres peuvent occuper jusqu'à 6 bits.

### 4.2.2 Gestionnaire de buffers

#### Motivations

Si la copie d'un octet n'est pas critique, celle d'un paquet reste problématique dans le cadre d'un matériel contraint tel que celui des capteurs. Pour éviter d'avoir à copier un paquet à chaque passage de paquet entre couche, nous choisissons de mettre en place un mécanisme simple de gestion de buffers. Celui-ci a pour objectif de fournir aux protocoles la possibilité de partager des buffers, contenant les paquets à traiter, sans avoir à les copier. Il doit également s'assurer que le buffer n'est plus utilisé par un autre protocole lorsqu'il va être réutiliser. Une autre contrainte est de pouvoir savoir quand l'identifiant du buffer a été entrer dans une file pour un traitement ultérieur. En effet, dans ce cas de figure, aucun utilisateur n'est en train de l'utiliser sans que cela ne signifie que le buffer est libre.

#### Mise en œuvre

La figure 4.2 montre les composants du gestionnaire de buffers.
Le gestionnaire utilise deux notions : celles de propriétaire et celle d'utilisateur. Seul un propriétaire peut demander à utiliser un buffer libre. L'API fournit les fonctions permettant de s'enregistrer comme propriétaires d'un nombre fixe de buffers. Si seule le rôle d'utilisateur est requis, l'enregistrement s'effectue comme propriétaire de 0 buffer. Un buffer est constitué de deux éléments : une structure de communication et le paquet proprement dit. La structure de communication permet aux couches d'y indiquer, entre autres, des informations concernant la source ou la destination du paquet. Les structures de communication peuvent différer selon les couches entre lesquelles s'effectuent ces communications. La taille réservée à la structure de

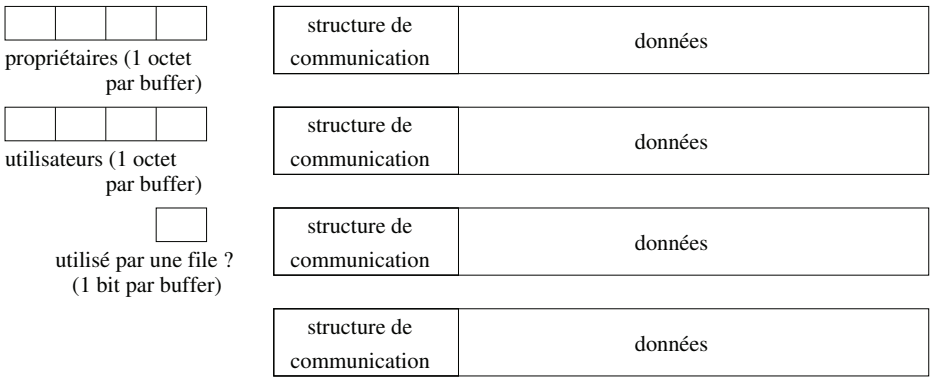



FIGURE 4.2 – *Gestionnaire de buffers. Il possède, outre les buffers proprement dit, un tableau de propriétaires (1 octet par buffer indiquant l'identifiant de son propriétaire), un tableau d'utilisateurs indiquant qui est en train d'utiliser le buffer et un tableau dans lequel un bit par buffer est réservé pour indiquer s'il est actuellement dans une file.*

communication dans le tableau servant de buffer est celle de la plus importante structure de communication de GOLIATH. Les identifiants des buffers sont stockés sur 6 bits, permettant ainsi de disposer de 32 buffers différents. Cette taille de 6 bits permet d'utiliser cette identifiant dans les messages de communication asynchrone. Ainsi, le format de l'octet utilisé pour la majorité des communications inter-couches sera de la forme :

(commande :2) (id buffer :6)

Les identifiants des propriétaires sont constitués d'un octet dans lequel un seul bit et à 1. Il est donc possible d'avoir 8 propriétaires de buffers au maximum. Ce format est choisi pour faciliter les opérations de marquage d'un buffer comme utilisé (un simple *et* logique suffit). Le gestionnaire dispose également d'un bit par buffer qui indique si celui-ci est actuellement utilisé par une file. De cette manière, le buffer ne sera pas libéré si il est en attente de traitement par un autre utilisateur. L'utilisation d'un buffer destiné à être passé à une autre couche adoptera généralement le schéma suivant :

1. Récupération de l'identifiant d'un buffer libre,
2. traitement divers,
3. remplissage de la structure de communication (destinataire ou source, longueur,...)
4. Passage à une autre couche :
   (a) indication au gestionnaire que le buffer va être passé dans une file,
   (b) message dans la file avec l'identifiant en paramètres du message.

En définitive, seul un octet est copié (deux fois) lors du passage d'un paquet d'une couche à une autre.

## 4.3 Couche physique et protocole MAC

### 4.3.1 Objectifs

La couche physique fournit un ensemble de primitives permettant d'accéder à l'interface radio. Elle permet de vérifier qu'un paquet entrant est bien formé (Somme de contrôle correcte, longueur du paquet correcte, destination). Elle est également responsable de la transmission effective des

données. La couche physique supporte par ailleurs le changement de puissance d'émission. Le protocole MAC qui sera utilisé par la PME Etineo sera 802.15.4. Celui implémenté dans Goliath est moins complexe et a pour unique objectif d'éviter les collisions. Il possède un mécanisme de backoff : si le nœud détecte qu'un autre nœud est en train d'émettre, il attend un temps aléatoire avant de retenter d'émettre. Un accusé de réception est requis pour tous les messages envoyés en unicast. Une frame contient la taille, le type (accusé de réception ou données), la destination et le numéro de séquence du message.

### 4.3.2 Mise en œuvre

#### Couche physique

La couche physique est la seule à ne pas être implémentée par une tâche. Elle n'utilise pas le gestionnaire de buffers et ne fournit qu'une API utilisée par la couche MAC. Elle permet d'initialiser une transmission. Une fois la transmission terminée, elle appelle la fonction enregistrée préalablement en callback. Une autre fonction fournie permet de vérifier qu'une frme reçue est correctement formée (la taille qu'elle indique faire est celle qu'elle fait effectivement, le nœud est bien la destination du paquet). Si c'est le cas, les en-têtes de la frame sont copiés dans une structure partagée avec la couche MAC. Cette structure contient la taille du paquet en octet, son type, l'adresse MAC de la source, l'adresse MAC de la destination, un numéro de séquence. La charge utile (*payload*) n'est pas copiée à ce moment là. Une autre fonction permet de récupérer ce payload. Lors de la réception d'un message, la couche physique empile un évènement dans la file d'attente de la couche MAC.

#### Couche MAC

La couche MAC est implémentée par une tâche unique et dispose d'une file d'attente lui permettant de recevoir des commandes d'autres couches. Une fois lancée, la couche MAC peut se trouver dans quatre états différents :

- `CAN_SEND_PKT` : état par défaut, elle peut recevoir ou envoyer des paquets ;
- `SENDING_PKT` : elle est en train d'envoyer un paquet et ne peut donc pas transmettre un autre paquet ou traiter un paquet entrant ;
- `WAITING_ACK` : elle est dans l'attente d'un accusé de réception pour un paquet envoyé en unicast ;
- `SENDING_ACK` : un paquet envoyé en unicast a été reçu, elle est en train d'envoyer l'accusé de réception correspondant.

Le format des messages acceptés dans la file d'attente (format de l'octet) est le suivant :

$$(\text{origine} : 2)(\text{paramètres} : 6).$$

L'origine indique si l'évènement provient de la couche supérieure (couche routage - paquet à envoyer) ou s'il s'agit d'un évènement interne (frame reçue ou prête à être envoyée).
L'algorithme 9 décrit la tâche correspondant à la couche MAC. Celle ci consiste a d'abord décider d'un délai d'attente sur la file d'évènement. Si le nœud est en attente d'un accusé de réception, le délai correspond au délai par défaut. Sinon, le délai choisit est le délai maximum. L'attente sur une file d'attente est bloquant et peut être interrompu pour deux raisons. La première est qu'un événement est arrivé dans la file d'attente (ou elle en contenait déjà un). Il peut s'agir d'une frame à envoyer ou d'une frame reçue. Si c'est une frame reçue et que celle ci est l'accusé de réception attendu, la fonction traitant le paquet libérera un sémaphore permettant à la fonction d'envoi de paquet de savoir que l'accusé de réception a été reçu. La couche MAC passe alors de l'état `WAITING_ACK` à l'état `CAN_SEND_PKT`.

```
state = CAN_SEND_PKT ;
while true do
    Se mettre en mode réception ;
    // décide du délai d'attente sur la file
    if state == WAITING_ACK then
        // En attente d'un accusé de réception
        delay = ACK_DELAY ;
    else
        delay = MAX_DELAY ;
    end
    // Attente sur la file, bloquant
    if évenement dans la file then
        if évènement interne then
            if frame reçue then
                traitement de la frame ;
            else
                // frame à envoyer
                envoyer la frame (appeler la couche physique) ;
            end
        else
            // évènement venant de la couche routage
            récupérer l'identifiant du buffer ;
            s'enregistrer comme utilisateur du buffer ;
            récupérer la structure de communication ;
            préparation du paquet et création d'un évenement interne "frame à envoyer" ;
            notifier au gestionnaire que le buffer n'est plus utilisé ;
        end
    else
        // Délai d'attente sur la file expiré
        if state == WAITING_ACK then
            // Accusé de réception non reçu
            retransmettre la frame ;
        end
    end
end
```

Algorithme 9 – *Tâche correspondant à la couche MAC.*

Lorsque la couche MAC reçoit un évènement de la couche routage, elle récupère le buffer et la structure de communication correspondant. Cette structure indique la destination du paquet et sa longueur.

## 4.4 Découverte de voisinage

Nous présentons dans cette section le mécanisme de découverte de voisinage implémenté dans Goliath ayant pour objectif de faire la distinction entre les nœuds vus occasionnellement et ceux pouvant être considérés comme fiables.

### 4.4.1 Objectifs

La découverte de voisinage permet de prendre connaissance des capteurs à portée de communication du nœud. Elle doit s'assurer que les liens pris en compte sont d'une qualité suffisante. En effet, comme avec un modèle de couche physique probabiliste il est difficile de déterminer à partir de quel moment il faut considérer un nœud comme voisin et l'utiliser dans les opérations de routage par exemple. Le medium étant instable, il est possible de ne recevoir que quelques paquets d'un nœud. Nous avons mis en place, dans Goliath, un processus de reconnaissance de voisinage illustré par la figure 4.4. La découverte s'effectue grâce à un ensemble de primitives et non par tâche dédiée, les primitives étant appelées par d'autres tâches telle que celle de la couche routage. Une de ces primitives permet d'envoyer des messages pour informer les autres nœuds de sa présence. Ces messages sont appelés *message HELLO*.
L'objectif est de ne considérer comme valide que les liens symétriques et d'une qualité suffisante en terme de probabilité de réception. Pour pouvoir déterminer si le lien est symétrique, il est nécessaire que chaque nœud inclus dans ses messages HELLO les nœuds dont il a connaissance. Nous proposons également un mécanisme d'estimation de la probabilité de réception basée sur les messages HELLO reçus. Il permet d'estimer le taux de réception par un nœud des $x$ derniers messages HELLO envoyés par un autre nœud en utilisant une file dans lesquels les temps des derniers messages reçus sont enregistrés. Il suffit alors de comparer le temps du premier élément de la file au temps courant pour estimer le taux de réception des paquets. Dans la sous-section suivante, nous expliquons plus en détail ces deux mécanismes (reconnaissance de voisinage et estimation du taux de réception) ainsi que la diffusion des messages HELLO.

### 4.4.2 Mise en œuvre

L'envoi des messages HELLO se fait à intervalles réguliers. Le format des messages HELLO est montré dans la figure 4.3. Le type indique que le paquet est un message HELLO. Le champ ADDR

| TYPE | ADDR | HOP_COUNT | FATHER | INTERVAL | TIME |
|---|---|---|---|---|---|
| NB NEIGHBORS | ID | RSSI | ..... | ID | RSSI |

Figure 4.3 – *Format des messages HELLO.*

correspond à l'adresse du nœud émetteur. Celui-ci indique également sa distance en nombre de sauts au puits (champ HOP_COUNT) et l'adresse de son père dans l'arbre (champ FATHER). Le champ INTERVAL correspond à l'intervalle auquel le nœud émetteur envoie les messages HELLO. Le champ TIME indique à quel temps le message a été envoyé (temps local de l'expéditeur).
Un nœud inconnu auparavant entre dans la catégorie des voisins potentiels à la réception d'un message HELLO si celui-ci est reçu avec une qualité suffisante. Cette qualité peut être estimée à

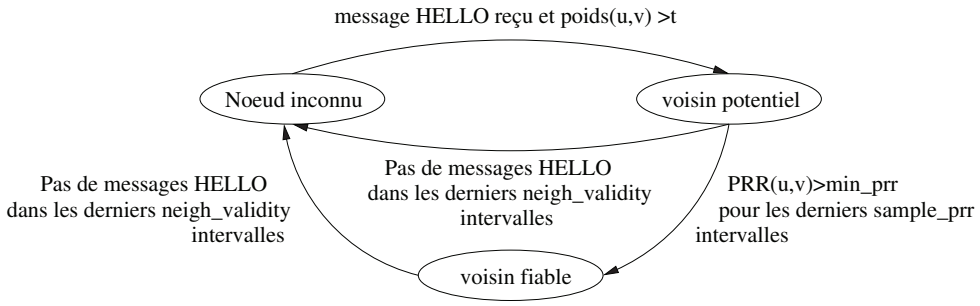



FIGURE 4.4 – *Mécanisme de reconnaissance de voisinage fiable.*

| ID | status | tree state | hop count | interval | next time entry | RSSI RX | RSSI TX | LQI RX | time | time buffer |
|---|---|---|---|---|---|---|---|---|---|---|

FIGURE 4.5 – *Entrée dans la table de voisinage contenant les informations concernant un nœud.*

l'aide de la puissance du signal reçu par exemple. Á partir de cet instant, une entrée est créée pour ce nœud dans la table de voisinage. Cette entrée contient l'identifiant du nœud, son statut (voisin potentiel, lien symétrique, ...), son état dans l'arbre, sa distance au puits, la puissance du signal reçu (RSSI RX), la puissance du signal émis (RSSI TX) et la date à laquelle l'entrée expire. A la création du nœud, son statut est celui de voisin potentiel. L'octet de statut contient plusieurs informations concernant ce nœud et le lien : si c'est un lien symétrique, Un voisin potentiel n'est pas utilisé pour les opérations de routage ou de contrôle de topologie. Il n'est pas non plus annoncé dans les messages HELLO. L'entrée lui correspondant dans la table de voisinage permet d'estimer le pourcentage de paquets reçus depuis ce nœud en se basant sur la réception des messages HELLO. En effet, l'entrée contient un tableau permettant d'estimer le pourcentage de messages HELLO reçu par rapport au nombre de messages que l'on aurait du recevoir. C'est la raison pour laquelle chaque nœud annonce dans son message HELLO à quel intervalle il les envoie. Ce tableau contient les temps auxquels les derniers messages HELLO ont été reçu (ce mécanisme est illustré par la figure 4.6). Pour déterminer si $p$ messages des $X$ derniers messages qui auraient du être reçus ont été effectivement était reçu, un tableau de taille $p \times X$ est créé. Le champ $next\_time\_entry$ permet de savoir à quel endroit ajouter la prochaine entrée dans ce tableau. Au départ, $next\_time\_entry = 0$. Cette variable est incrémentée à chaque réception d'un message HELLO envoyé par ce nœud. Lorsque le buffer est plein, la date du plus ancien message HELLO reçu de ce nœud ($oldest\_date$) est comparée à la date courante ($current\_date$). Si :

$$oldest\_date > current\_date - X \times interval$$

alors cela signifie que le ratio de messages HELLO reçu est suffisant et le voisin potentiel devient un voisin fiable. Il est alors utilisé pour le routage et est annoncé dans les messages HELLO. Si la condition n'est pas vérifiée, le nœud reste un voisin potentiel et à chaque réception d'un de ses messages HELLO, la condition est revérifiée pour savoir s'il peut passer dans l'état de voisin fiable.
Si aucun message HELLO n'a été reçu pendant $neigh\_validity$ intervalles consécutifs, un voisin potentiel ou un voisin fiable redevient un nœud inconnu.
Les primitives proposées pour la découverte de voisinage sont appelées par la tâche correspondant au protocole de routage que nous allons détaillé dans la section suivante.

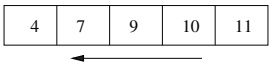



FIGURE 4.6 – *Buffer permettant d'estimer le ratio de paquets reçus pour chaque voisin. Il contient le temps auquel ont été reçus les derniers messages HELLO du nœud correspond. Lorsque le buffer est plein, la première entrée (donc la plus ancienne) est comparée au temps courant. Supposons que le nœud envoie ses paquets à un intervalle de* 1 *et que le seuil pour être considéré comme fiable est de 5 messages reçus parmi les 10 derniers envoyés. Si le temps courant est de* 13 *par exemple, le nœud est considéré comme voisin fiable (*$5 > 13 - 10 \times 1$*).*

## 4.5 PROTOCOLE DE ROUTAGE

### 4.5.1 Objectifs

Le protocole de routage a la responsabilité d'acheminer les messages d'une source vers une destination en utilisant éventuellement d'autres nœuds comme relais. Dans GOLIATH, le routage est effectué à l'aide d'un arbre : chaque nœud sélectionne un père auxquels il envoie les messages destinés au puits. On parle de routage ascendant. Lorsque les messages sont originaires du puits et destinés à un ou des nœuds du réseau, on parle de routage descendant. Ce routage se doit de supporter la dynamicité du réseau : apparition ou disparition de nœuds, instabilité d'un lien qui était jusqu'alors fiable, etc. Pour ceux faire, la couche routage assure deux rôles : la construction et la maintenance de l'arbre et le routage proprement dit. Nous choisissons un routage saut à saut : chaque nœud chargé de relayer un paquet en choisit le prochain saut. Dans notre cas, le prochain saut sera le père du nœud dans l'optique d'un routage ascendant. La construction et la maintenance de l'arbre se base sur les messages HELLO, dans lesquels le nœud indique également sa distance en nombre de sauts au puits et l'identifiant de son père (voir figure 4.3).

### 4.5.2 Mise en œuvre

#### Construction et maintenance de l'arbre

Le protocole de routage utilisé se base sur la construction d'un arbre dont le puits est la racine. Un nœud peut se trouver dans quatre états différents :

- `ORPHAN` : le nœud n'est pas attaché à l'arbre. Il s'agit de l'état dans lequel se trouve un nœud non puits lorsqu'il rejoint le réseau. Il peut également se trouver dans cet état si il a perdu la connexion avec son pére ;
- `ATTACHING` : le nœud a entamé la procédure d'attachement à un autre nœud du réseau qui est lui-même attaché ;
- `ATTACHED` : la procédure d'attachement a abouti, le nœud dispose d'un pére et peut maintenant accepté les requêtes d'autres nœuds lui demandant d'être leur père. Le puits se trouve toujours dans cet état ;
- `DETACHING` : le nœud vient de perdre la connexion avec son père et doit donc retrouver un autre père. Il passera dans l'état `ORPHAN` lorsque tout ses fils seront eux-mêmes dans l'état `ORPHAN`.

Chaque nœud inclut dans le message HELLO qu'il envoie régulièrement sa distance en nombre de sauts et l'identifiant de son père. S'il n'est ni attaché à l'arbre ni en procédure d'attachement, ces valeurs sont `UNKNOWN_HOP_DIST` et `UNKNOWN_NODE` pour la distance en nombre de sauts et l'identifiant du père respectivement. Ces deux valeurs permettent aux nœuds voisins de connaître l'état du nœud. La tableau 4.7 montre l'état d'un nœud en fonction de la connaissance de ces deux valeurs. Une distance en nombre de sauts ainsi qu'un père connu indique que le nœud est attaché à l'arbre. Si seule la distance au puits est connue, cela signifie que le nœud a perdu le lien avec son père (celui ci est devenu asymétrique, ou le nœud est maintenant un nœud inconnu) et qu'il

| père \ distance | connue | inconnue |
|---|---|---|
| **connu** | ATTACHED | ATTACHING |
| **inconnu** | DETACHING | ORPHAN |

FIGURE 4.7 – *Déduction de l'état d'un nœud en fonction des informations contenues dans le message HELLO.*

| TYPE | ADDR | FATHER |
|---|---|---|

(a) Requête d'attachement

| TYPE | ADDR | CHILD |
|---|---|---|

(b) Accusé de réception d'attachement

FIGURE 4.8 – *Requête et accusé de reception d'attachement.*

est par conséquent dans l'état `DETACHING`. Inversement, s'il connaît son père mais pas sa distance en nombre de sauts au puits, il est dans l'état `ATTACHING`. Il est dans l'état `ORPHAN` si ces deux données sont inconnues. Si un nœud est dans l'état `ORPHAN` et qu'un de ses voisins fiables avec lequel il a un lien symétrique est dans l'état `ATTACHED`, il entame une procédure d'attachement à ce nœud. Celle ci se déroule en deux étapes. Dans un premier temps, il envoie une requête d'attachement à ce nœud et passe dans l'état `ATTACHING`. L'identifiant de son père est maintenant celui de ce voisin, mais la distance en nombre de sauts est toujours inconnue. Le format des paquets de requête d'attachement est montré sur la figure 4.8(a). Le nœud recevant cette requête vérifie qu'il possède un lien symétrique avec l'expéditeur. Si c'est le cas, il envoie une accusé de réception positif au nœud qui est maintenant son fils dans lequel il confirme sa distance au puits (figure 4.8(b)). A la réception de cet accusé, le nœud passe dans l'état `ATTACHED` et complète sa distance au puits. Un délai d'expiration est associée à la requête. Si celui ci expire et qu'aucun accusé n'a été reçu, le nœud repasse dans l'état `ORHAN`. Comme mentionné précédemment, si un nœud perd le lien symétrique avec son père, il passe dans l'état `DETACHING`. Il retournera dans l'état `ORPHAN` lorsque tous ses fils ne seront plus dans l'état `ATTACHED` avec lui comme père. Ce processus permet de s'assurer que l'on ne crée pas de *boucle*. En effet, imaginons que le nœud pas directement dans l'état `ORPHAN`. Dans ce cas là, il est possible qu'il tente de s'attacher à l'un de ses fils et ainsi avoir une distance en nombre de sauts au puits erronée.

### Routage dans l'arbre

Le routage dans l'arbre est simple et optimisable. Un nœud désirant envoyé un message au puits (cas le plus probable dans un réseau de capteurs) le fait en envoyant le paquet à son père qui se chargera de le router et ainsi de suite jusqu'à ce que le puits le reçoive. Si un nœud n'est pas dans l'état `ATTACHED`, c'est-à-dire qu'il n'est pas rattaché à l'arbre, il envoie son paquet en broadcast. Les nœuds le recevant pour la première fois le retransmettent. Le routage descendant dans l'arbre s'effectue également en broadcast. La figure 4.9 donne format des paquets de données. L'en-tête d'un paquet de données comprend le type du paquet (données en l'occurrence), sa taille, les adresses source et destination et un numéro de séquence. A la réception d'un message, l'adresse de l'expéditeur et le numéro de séquence sont stockés dans une table dédiée. Celle ci est utilisée pour savoir si le message a déjà été traité.

La couche routage est implémentée par une unique tâche, comme la couche MAC. Cette tâche est en fait une boucle infinie qui attend l'arrivée d'items dans la file d'attente lui permettant de recevoir des messages des couches supérieure et inférieure, ou d'elle-même. Si l'action demandée est demandée par une autre couche, l'octet de poids fort est à 0. Dans ce cas, le bit suivant


FIGURE 4.9 – *Format des paquets de données. Il contient le type indiquant qu'il s'agit d'un paquet de données, sa taille, la source et la destination ainsi qu'un numéro de séquence.*

dépendra du fait que ce soit la couche supérieure ou la couche inférieure qui demande le traitement du paquet. Dans les deux cas, les bits suivant servent à indiquer l'identifiant du buffer où sont stockés le paquet et la structure de communication inter-couche. Si l'octet dans la file provient de la couche inférieure, c'est-à-dire s'il s'agit d'un paquet provenant de la couche MAC, il est traité selon son type. Si c'est un paquet de données, il est routé de la manière décrite précédemment.

## 4.6 PROTOCOLES DE TRANSPORT

### 4.6.1 Objectifs

La couche transport est responsable de la fiabilité des transmissions de données de bout en bout. Nous avons donc pour objectifs de fournir des protocoles de transport compatibles avec les contraintes liées aux réseaux de capteurs. Les principaux problèmes rencontrés sont le faible espace mémoire disponible sur les capteurs ainsi que l'aspect onéreux des communications. TCP se révèle trop lourd pour des systèmes ainsi contraints.
GOLIATH dispose de deux protocoles de transport léger. Le premier, DTP (*Datagram Transport Protocol*) est conçu pour envoyer des messages dont la perte n'est pas critique pour l'application. Un exemple typique est le relevé régulier de température. Si un relevé est perdu, l'application et l'utilité du réseau n'en sont que modérément affectées. DTP ne fournit que la fragmentation et l'envoi des fragments sans accusé de réception. L'avantage de cette solution est qu'elle n'engorge pas le réseau avec les accusés de réception et ne nécessite pas à la couche transport de conserver le fragment qu'elle vient d'envoyer dans l'attente d'un accusé de réception.
ETP (*End-to-end Transport Protocol*) est utilisé dans le cas où la perte d'un message peut se révéler préjudiciable pour l'application. Un exemple serait l'envoi d'une alerte en cas de température trop élevée détectée par un capteur. ETP s'assure donc que le message ait été correctement réceptionné en demandant un accusé de réception. Si le fragment n'a pas été accusé, il est réexpédier. Cette solution nécessite de conserver les fragments déjà envoyés. Dans un soucis d'économie de mémoire et de gestion, un seul fragment en vol est autorisé.

### 4.6.2 Mise en œuvre

La structure d'un segment ETP ou DTP est montrée sur la figure 4.10. L'en-tête d'un segment comprend un flag indiquant quel protocole est utilisé, un numéro de fragment, la taille de ce fragment en octets et un numéro de séquence. Ce dernier est le même pour tous les fragments d'un même message.

#### DTP : protocole de transport en mode datagramme

DTP (*Datagram Transport Protocol*) est le protocole de transport en mode datagramme fournit par GOLIATH. Il est utilisé pour les messages dont la perte n'est pas critique. Par exemple, si le capteur a pour tâche d'effectuer des mesures sur son environnement à intervalles réguliers et de transmettre ces données au puits, la perte d'un de ces messages n'est pas obligatoirement

```
while true do
    Calculer la date du prochain envoi de message HELLO ;
    if date du prochain envoi passée then
        envoi du message HELLO ;
    else
        // Attente sur la file jusqu'à la date d'envoi du prochain message
           HELLO
        if événement sur la file then
            if message d'une autre couche then
                if message couche supérieure then
                    récupérer l'identifiant du buffer ;
                    récupérer la structure de communication ;
                    // route le paquet
                    if le message n'est pas pour le puits then
                        if le message est pour un nœud voisin avec lien symétrique then
                            envoyer directement le message au nœud ;
                        else
                            envoyer le message en broadcast ;
                        end
                    else
                        // le message est à destination du puits
                        if le nœud est dans l'état ATTACHED then
                            envoi du message au père ;
                        else
                            envoi du message en broadcast ;
                        end
                    end
                else
                    // message couche inférieure
                    if c'est la première fois que l'on voit ce message then
                        if le nœud est la destination then
                            transmettre à la couche supérieure (ajouter un événement dans sa
                            file) ;
                        else
                            router le paquet (même règles que précédemment) ;
                        end
                    end
                end
            else
                // commande interne - envoi message HELLO
                envoyer le message HELLO ;
            end
        else
            // délai d'envoi du message HELLO passé, pas de message sur la
               file
            envoyer le message HELLO ;
        end
    end
end
```

Algorithme 10 – *Tâche correspondant à la couche routage.*


FIGURE 4.10 – *Format des fragment de la couche transport. Le premier octet indique s'il s'agit d'un fragment ETP ou d'un fragment DTP. Il permet également de savoir si ce fragment est le dernier du message. Le* `FRAG NB` *correspond au numéro du fragment. S'il ne correspond pas à celui attendu, il est droppé et aucun accusé de réception n'est envoyé. Le numéro de séquence est défini pour l'ensemble du message, chaque fragment d'un même message possède le même numéro de fragment.*

problématique. Toutefois, si le nœud est rattaché à l'arbre, il dispose des accusés de réception de saut en saut fournis par la couche MAC.

### ETP : protocole de transport de bout en bout

ETP (*End-to-end Transport Protocol*) est le protocole de transport de bout en bout. Il utilise un mécanisme d'accusé de réception. En cas de détection de perte, le paquet est retransmis. Pour économiser de l'espace mémoire et simplifier la gestion des flux, un seul paquet en vol est autorisé. La perte d'un paquet de bout en bout est géré par un mécanisme de temporisation et de retransmission. La temporisation s'effectue par l'estimation du *Round Trip Time* (RTT) similaire à celui mis en place dans TCP. Il s'agit de l'estimation du temps que met un paquet à faire l'aller-retour entre l'expéditeur et le destinataire. Lorsque ETP envoie un fragment, il conserve l'adresse du destinataire, le numéro de séquence du message et le numéro du fragment. Ces informations seront utilisées pour vérifier que l'accusé de réception éventuellement reçu correspond au fragment envoyé. Si l'accusé n'est pas reçu dans le temps imparti (le RTT), celui-ci est doublé et le fragment rééxpédié. Le nœud dispose d'un buffer dont chaque entrée correspond au RTT associé aux destinataires les plus récents. Si l'accusé de réception est reçu avant l'expiration du RTT, celui ci est mis à jour comme suit :

$$RTT_{new} = RTT_{old} \times \alpha + RTT \times (1 - \alpha),$$

où $RTT$ représente le $RTT$ calculé pour ce fragment, $RTT_{old}$ le RTT avant mise à jour et $RTT_{new}$ la valeur du RTT après mise à jour. Le paramètre $\alpha$ permet de régler l'influence de la non réception d'un accusé de réception sur la nouvelle valeur du RTT. Le RTT par défaut est également mis à jour à cette occasion.

La couche transport est également implémentée par une tâche unique attendant que des données arrivent sur la file dont elle est propriétaire. Elle propose également deux fonctions pouvant être utilisées par l'application : une pour l'envoi de données (`send()`), l'autre pour la réception (`receive()`). La couche transport peut se trouver dans trois états :

- `TP_READY` : il s'agit de l'état normal de la couche transport ;
- `TP_RCV_DTP` : la couche transport est en train de recevoir un message fragmenté et la réception se fait en utilisant DTP ;
- `TP_RCV_ETP` : la couche transport est en train de recevoir un message fragmenté et la réception se fait en utilisant ETP.

Lorsque la fonction `send()` est appelée, le paquet est fragmenté au fur et à mesure et un événement est placé dans la file d'attente à chaque envoi de fragment. S'il ne s'agit pas du dernier fragment et que l'envoi se fait en utilisant DTP, le nœud reste dans l'état `TP_RCV_DTP`. Il adopte le même comportement lors de la réception d'un message en utilisant ETP. Si le fragment arrivant ne correspond pas à celui attendu, le message est ignoré si DTP est utilisé. Si ETP est utilisé, la source du message devrait normalement renvoyer le ou les fragments perdus étant donné qu'ils n'auront pas été acquittés.
L'algorithme 11 correspond à la tâche de la couche transport. Une vue globale de GOLIATH incluant les files d'attentes est montrée sur la figure 4.11.

```
tp_state = TP_READY while true do
    // calcul du timeout avant d'arrêter d'attendre un événement
    if tp_state == TP_READY then
        delay = MAX_DELAY ;
    else
        // le nœud est dans l'état TP_RCV_DTP ou TP_RCV_ETP - calcul le délai par rapport au RTT estimé
        timeout = last_pkt_time + rtt ;
        if timeout < now then
            // délai dépassé
            delay = 0 ;
        else
            delay = timeout − now ;
        end
    end
    if événement dans la file avant expiration de delay then
        récupérer l'identifiant du buffer et la structure de communication ;
        switch tp_state do traitement selon l'état du nœud
            case TP_READY
                if accusé de réception then
                    vérifier qu'il s'agit de l'accusé de réception en attente ;
                    if accusé attendu then
                        // le sémaphore est rendu et peut être pris par la fonction send()
                        rendre le sémaphore ;
                    end
                    break ;
                end
                // ce n'est pas un accusé de réception
                if frag_number ≠ 0 then
                    // le nœud a manqué le premier fragment, message droppé
                    break ;
                end
                if paquet ETP then
                    envoi accusé de réception ;
                end
                if dernier fragment then
                    // le message ne fait qu'un fragment, le traitement est terminé
                    ajouter événement dans la file de l'application ;
                    break ;
                else
                    if paquet ETP then
                        tp_state = TP_RCV_ETP
                    else
                        tp_state = TP_RCV_DTP
                    end
                end
            endsw
            case TP_RCV_DTP
                if il ne s'agit pas du prochain fragment attendu - ignorer then
                    tp_state = TP_READY ;
                    envoi événement d'erreur sur la file d'application ;
                    break ;
                end
                if dernier fragment du message then
                    tp_state = TP_READY ;
                else
                    mise à jour des informations concernant le prochain fragment attendu ;
                end
                ajout d'un événement dans la file de l'application ;
            endsw
            case TP_RCV_ETP
                if il ne s'agit pas du prochain fragment attendu then
                    // le prochain devrait être renvoyé par la source
                    break ;
                end
                envoi de l'accusé de réception ;
                if dernier fragment du message then
                    tp_state = TP_READY ;
                else
                    mise à jour des informations concernant le prochain fragment attendu ;
                end
                ajout d'un événement dans la file de l'application ;
            endsw
        endsw
    else
        // le timeout a expiré et il n'y a pas eu d'événement - abandon de la réception
        tp_state = TP_READY ;
        envoi événement d'erreur sur la file d'application ;
    end
end
```

Algorithme 11 – *Tâche correspondant à la couche transport. À chaque fois qu'un événement est envoyé sur la file de l'application, il est traité la fonction receive(). Chaque buffer utilisé pour la copie des données l'est avec l'identifiant de l'application.*

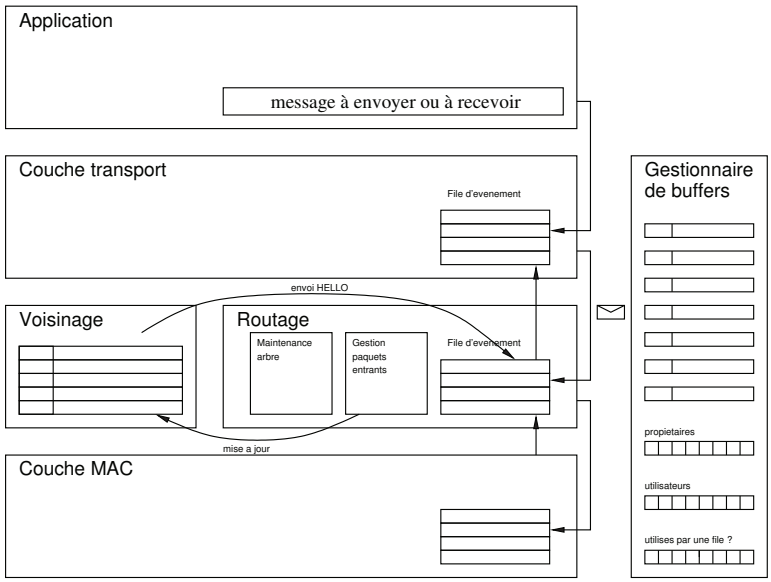



FIGURE 4.11 – *Vue d'ensemble de* GOLIATH *incluant les files d'événement. Lorsque l'application désire envoyé un message, elle appelle une primitive de la couche transport. Celle-ci ajoute un événement dans la file d'attente de la couche transport à chaque fragment devant être envoyé. La couche transport procède de même avec la couche routage qui transmet l'événement à la couche MAC.*

| | | |
|---|---|---|
| CPU | Type<br>Fréquence | TI MSP430-1611 (16 bits)<br>8 MHz |
| Mémoire | mémoire flash programmable<br>RAM<br>Mémoire flash | 48KB<br>10KB<br>1MB |
| radio | Type<br>bandes de fréquence<br>puissances d'émission | TI CC1101<br>315/433/868 and 915 MHz<br>-30 dBm to +10dBm |

FIGURE 4.12 – *Caractéristiques matérielles des nœuds WSN430 utilisés pour l'évaluation de* GOLIATH. *Nous utilisons la fréquence de* 868*MHz*

## 4.7 ÉVALUATION

L'évaluation de GOLIATH se fait dans le cadre d'une utilisation en intra-bâtiment. L'objectif est de tester la stabilité de l'arbre et le taux de réception des messages en utilisant la puissance de transmission maximale. Cette évaluation s'effectue en utilisant les capteurs sans fil WSN430 (version 1.3). Ceux-ci sont équipés du microprocesseur 16 bits MSP430 cadencé 8Mhz(voir tableau 4.12. Ils disposent de 10 Ko de RAM et de 48 Ko de mémoire flash programmable ainsi que d'une mémoire flash d'1 Mo. Plusieurs capteurs sont fournis permettant de mesurer la température et la luminosité. Chaque capteur possède un identifiant unique fournit matériellement. Le tableau 4.13 montre les différents paramètres utilisés dans GOLIATH pour la phase de test. Le nombre de buffers disponibles au total est de 10, dont 3 pour la couche MAC, 1 pour la couche routage, et 3 pour la couche transport. Lors de l'envoi d'un paquet en unicast, la couche MAC attend pendant 150ms l'accusé de réception avant de le ré-émettre (au maximum 5 fois). Le nombre maximum de voisins pour un nœud est de 10 et le nombre maximum d'entrées dans la table mémorisant les messages déjà traités par la couche routage est de 15. L'envoi de messages HELLO s'effectue toutes les 10s. Le taux de messages reçus nécessaires avant qu'un voisin soit considéré comme fiable est de 0.6 sur les 10 derniers paquets.
Les dix capteurs sont disposés de manière arbitraire (voir figure 4.16). Chacun d'eux envoie au

| Paramètre | Valeur |
|---|---|
| fréquence | 868MHz |
| NB_BUFFERS | 10 |
| MAC_NB_BUFFERS | 3 |
| ACK_DELAY | 150ms |
| MAX_RETRIES | 5 |
| NET_NB_BUFFERS | 1 |
| MAX_NB_NEIGHBORS | 10 |
| HELLO_MSG_INTERVAL | 10s. |
| ATTACH_REQ_TIMEOUT | 2s. |
| MIN_PRR | 0.6 |
| SAMPLE_FOR_PRR | 10 |
| SEEN_MSG_MAX_NB_ENTRIES | 15 |
| TP_NB_BUFFERS | 3 |

FIGURE 4.13 – *Paramètres utilisés lors de l'évaluation de* GOLIATH

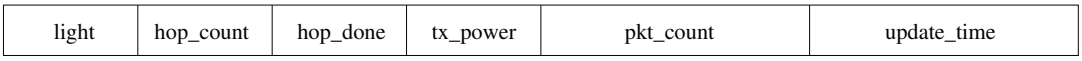



FIGURE 4.14 – *Paquet envoyé lors de l'évaluation de* GOLIATH. *Il contient la luminosité mesurée, la distance du nœuds au puits en nombre de sauts (`hop_count`), le nombre de sauts parcourus par le paquet (`hop_done`), la puissance de transmission utilisée par la source (`tx_power`), le nombre de paquets envoyés (`pkt_count`) et le temps local de dernière mise à jour de la table de voisinage (`update_time`).*

puits toutes les 10 secondes la luminosité qu'il mesure durant 2 heures en utilisant le protocole de transport DTP. Le choix de DTP a été fait pour pouvoir estimer le taux de réception sans possibilité de retransmission de bout en bout. Le message que le nœud envoie, dont le format est montré sur la figure 4.14, contient également d'autres informations de nature topologique. Ces informations sont la distance en nombre de sauts au puits, la puissance de transmission utilisée par le nœud source, le nombre de paquets envoyés au total par ce nœud et le temps auquel le paquet a été envoyé. De plus, à chaque fois que la couche routage décide de relayer un paquet, elle incrémente le nombre de sauts parcourus par celui-ci.

De plus, toutes les 30 secondes chaque nœud enregistre dans sa mémoire flash d'autres informations de topologie. Ces informations sont le temps de l'enregistrement, le nombre de paquets envoyés, la puissance de transmission utilisée, l'identifiant du pére, la distance en nombre de sauts au puits, le nombre de voisins connus (potentiel ou fiable) et pour chacun d'eux l'identifiant, le statut, le RSSI et le LQI correspondant. Le format des données sauvegardées est montré sur la figure 4.15.

**Distance au puits :** La figure 4.17 donne la distance en nombre de sauts au puits. Sont également donnés le pourcentage de temps passé à cette distance pour les nœuds étant principalement à

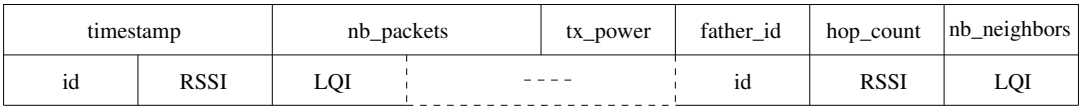



FIGURE 4.15 – *Données sauvegardées en flash lors de l'évaluation de* GOLIATH. *La structure contient le temps local de la sauvegarde (`timestamp`), le nombre de paquets envoyés à cet instant, la puissance de transmission utilisée, l'identifiant du père et la distance au puits, le nombre de voisins et pour chacun d'eux l'identifiant et le RSSI et le LQI en réception.*

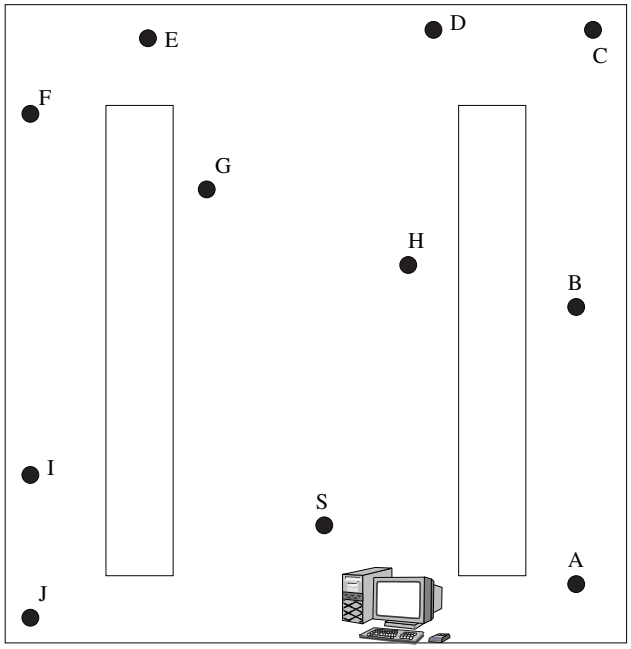



FIGURE 4.16 – *Répartition des capteurs. Le puits est désigné par la lettre S. Tous les nœuds utilisent la puissance de transmission maximale(+10dBm).*

cette distance. Neuf des dix nœuds se trouvent à portée du puits, et passe en moyenne 76.6% du temps à cette distance.

**Degré moyen :** Le degré des nœuds est d'en moyenne 7.3.

**Pourcentage de paquets reçus :** La figure 4.18 donne, pour chaque nœud, le pourcentage de paquets acheminés jusqu'au puits. On constate que pour $C$,$D$,$G$ et $H$ les performances sont moins bonnes que pour les autres nœuds dont le pourcentage de paquets correctement acheminés dépasse les 90%.

**Exemple du nœud $J$ :** La figure 4.19 montre à quelles distances en nombre de sauts du puits s'est trouvé le nœud $J$. Il était majoritairement à un saut (plus de 75% du temps). Étrangement, lorsqu'il se trouvait à deux sauts, il choisissait $H$ comme père dans l'arbre. Son état au niveau arbre s'est réparti comme suit :

| **état** | **% de temps** |
|---|---|
| `ORPHAN` | 3.95 |
| `ATTACHING` | 3.16 |
| `ATTACHED` | 90.51 |
| `DETACHING` | 2.37 |

Le nœud se trouve, comme attendu, dans l'état `ATTACHED` la majeure partie du temps, démontrant ainsi une certaine stabilité.

## 4.8 Conclusion

Dans ce chapitre nous avons présenté Goliath, une pile de communication pour réseaux de capteurs. Dans l'évaluation mise en place, Goliath donne de bonnes performances, avec un taux de réception par le puits d'un peu plus de 75%. Des améliorations sont bien évidemment envisageables. Par exemple, il serait possible d'utiliser un algorithme tel que TAP [28] pour décider de la fréquence des envois des messages HELLO, notamment en cas de mobilité des nœuds.

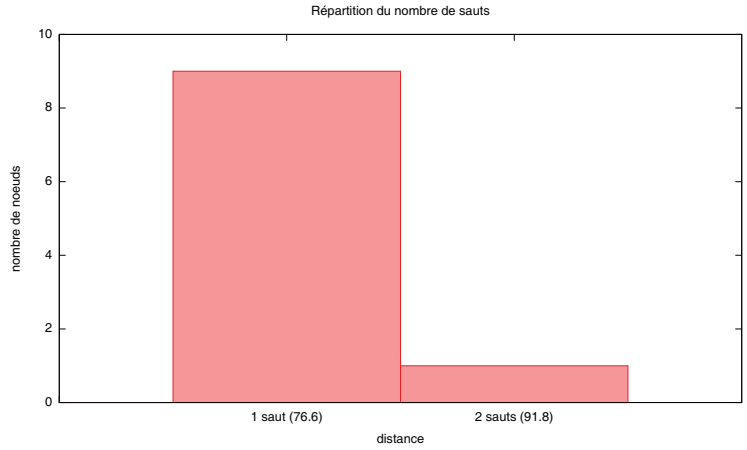



FIGURE 4.17 – *Répartition des distances en nombre de sauts du puits. Les nombres entre parenthèses donnent le pourcentage de temps moyen passé à cette distance en nombre de sauts par les nœuds se trouvent à cette distance la majorité du temps.*

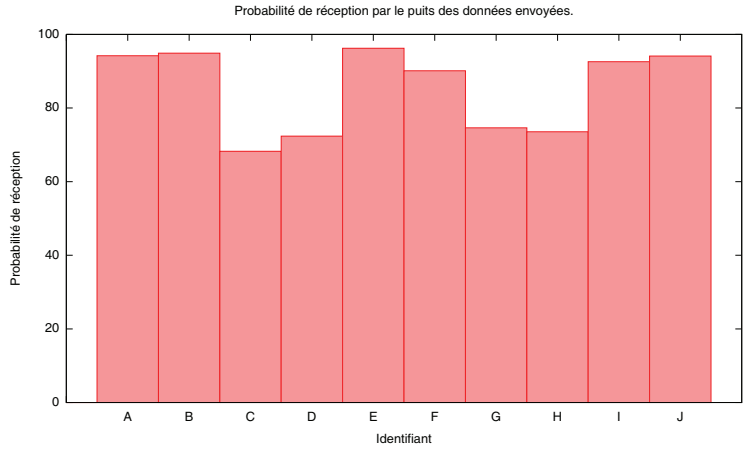



FIGURE 4.18 – *Pourcentage de réception par le puits des messages envoyés par les nœuds.*


FIGURE 4.19 – *Répartition de la distance en nombre de sauts du puits pour le nœud J. Les nombres entre parenthèses donnent le pourcentage de temps moyen passé à cette distance en nombre de sauts.*

Le routage dans l'arbre n'est également pas optimisé. Pour l'instant cela n'est pas critique étant donné le nombre peu élevé de nœuds. Des améliorations sont possibles en utilisant un mécanisme de nommage qui pourrait permettre de traiter les cas du routage descendant sans utiliser de diffusion et de faciliter les envois de messages entre nœuds. C'est le mécanisme adopté dans ZigBee. D'autres approches existent, telles que HECTOR [8].
Concernant la couche transport, si plusieurs applications sont présentes dans le capteur et qu'elles envoient des données régulièrement, il pourrait être intéressant de fournir un service de sockets. Cependant, ce mécanisme demande des ressources conséquentes en terme de mémoire.

# Contrôle de topologie dans Goliath

5

Sommaire



Dans ce chapitre, nous présentons l'implémentation de la réduction de graphe des voisins relatifs (RNG - *Relative Neighborhood Graph*) appliqué dans Goliath. Le graphe des voisins relatifs a été retenu car il ne nécessite qu'une informations sur le 2-voisinage. Il permet d'obtenir un degré relativement faible tout en permettant d'éliminer les liens les moins fiables. Une fois le graphe RNG calculé, chaque nœud adapte sa puissance de transmission de manière à atteindre son voisin le plus éloigné (par rapport à la fonction de distance utilisée) dans ce graphe. L'objectif est de diminuer, autant que possible, la puissance de transmission de chaque nœud. Le tableau 5.1 montre la consommation énergétique à différentes puissances de transmission pour une fréquence de 868MHz. L'énergie dépensée peut varier du simple au triple. Pour illustrer notre démarche, nous commençons par étudier la relation entre l'indicateur de puissance reçu et l'indicateur de qualité de lien d'une part, et le ratio de paquets reçus d'autre part. Nous présentons ensuite la méthode utilisée pour adapter le RNG dans le cadre d'une utilisation sans système de positionnement et en considérant la qualité du lien comme distance. Nous discutons ensuite des résultats obtenus lors de son évaluation.

| **Puissance (dBm)** | -30 | -20 | -15 | -10 | 0 | 5 | 7 | 10 |
|---|---|---|---|---|---|---|---|---|
| **Consommation (mA)** | 12.0 | 12.6 | 13.3 | 14.5 | 16.8 | 19.9 | 25.8 | 30.0 |

FIGURE 5.1 – *Consommation énergétique pour différentes puissances de transmission à une fréquence de 868MHz. L'énergie dépensée varie du simple au triple [9].*

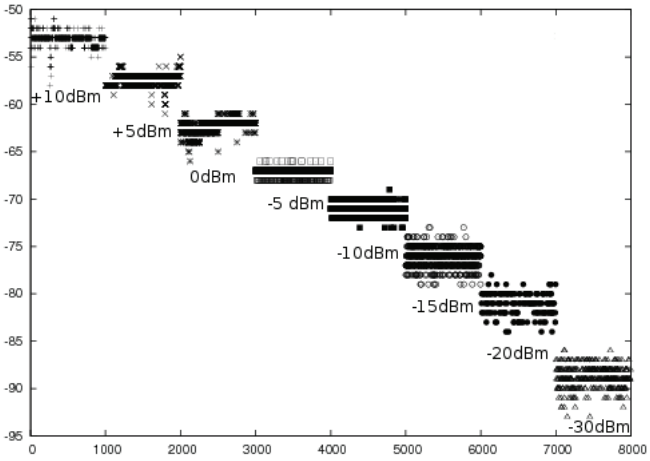



FIGURE 5.2 – *Évolution du RSSI avec le temps et la puissance de transmission.*

## 5.1 Corrélation RSSI/LQI et PRR

Nous étudions dans cette section l'évolution du RSSI, du LQI et du PRR lors de l'utilisation de différentes puissances de transmission.

### 5.1.1 Présentation du RSSI et du LQI

Le LQI (*Link Quality Indicator*) est une métrique proposée par de nombreux chips radio dont le $CC1101$. Il indique la difficulté du décodage du signal reçu. Il compare donc le signal reçu au signal qui aurait été reçu dans des conditions idéales. Un LQI faible représente un signal de meilleure qualité qu'un LQI plus élevé.
Le RSSI (*Received Signal Strength Indicator*) est un indicateur de la puissance du signal reçu, mais pas de sa qualité. Cependant, un signal fort souffre moins du bruit, et donne donc souvent un signal de qualité au niveau du receveur. Il peut donc exister quatre à cinq cas extrêmes concernant le RSSI et le LQI [42] :

- Un signal faible et peu de bruit donnent un RSSI faible mais un LQI élevé. Le message est difficile à décoder ;
- S'il y a absence totale de bruit, le LQI sera également bas,
- Un bruit fort couvrira le signal, donnant un LQI faible, alors que le RSSI sera élevé ;
- Si le signal est fort et le bruit aussi, on peut obtenir un RSSI ainsi qu'un LQI élevés ;
- Le signal peut être fort au point de saturer le receveur, qui aura un RSSI élevé mais un LQI important également.

Nous étudions tour à tour la corrélation entre le RSSI et la puissance d'émission, le LQI et la puissance d'émission ainsi qu'entre le PRR et la puissance d'émission. Les tests sont effectués en intérieur, dans un environnement perturbé.

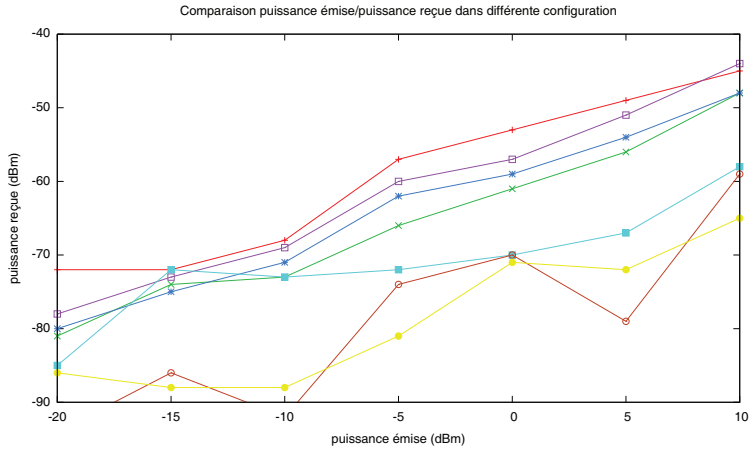



FIGURE 5.3 – *Évolution de la puissance reçue en fonction de la puissance émise. Ces mesures sont prises à différentes distances variant* 1 *à* 15 *mètres. La pente reste la même.*

### 5.1.2 Étude de la corrélation RSSI/puissance d'émission

La figure 5.2 montre l'évolution du RSSI en fonction de la puissance de transmission. Les résultats sont obtenus en ayant deux nœuds : un émetteur et un receveur. Le nœud recevant les paquets d'une taille de 60 octets enregistre le RSSI et le LQI qui lui sont associés. Pour le test dont les résultats sont montrés sur la figure 5.2, 1000 paquets de suite sont envoyés à chaque puissance de transmission. Chaque paquet reçu et le RSSI correspondant sont montrés dans la figure 5.2. Elle confirme la forte corrélation entre la puissance de transmission et le RSSI. Plus la puissance de transmission diminue, moins le nombre de paquets reçus est important et le RSSI correspondant est faible.

La figure 5.3 montre le RSSI moyen obtenu dans différentes configurations (distance entre l'émetteur et le récepteur variant de 1 à 15 mètres) et différentes puissances de transmission. Nous omettons volontairement les distances sur la figure étant donné qu'elles n'ont que peu d'importance. Nous ne nous intéressons pas à l'évolution du RSSI en fonction de la distance. Seule l'évolution du RSSI en fonction de la puissance de transmission importe. On constate que pour des valeurs de RSSI supérieures à $-70$dBm il est possible de déterminer comment elles évoluerons en fonction de la puissance de transmission.

### 5.1.3 Étude de la corrélation LQI/puissance d'émission

En théorie, le LQI devrait augmenter avec la diminution de la puissance de transmission pour un environnement constant. Les résultats de la figure 5.2 ont été obtenus de la même manière que ceux de la figure 5.4 :1000 paquets de suite sont envoyés à chaque puissance de transmission et on relève pour chaque paquet reçu le LQI correspondant. Il y a effectivement corrélation entre le LQI et la puissance d'émission. Le LQI a tendance à "s'éparpiller" au fur et à mesure que la puissance d'émission diminue, dénotant un fort écart-type. Lorsque le nombre de paquets reçu diminue, on assiste à l'apparition de valeurs de LQI discriminantes. Ainsi, une valeur de LQI supérieure à 200 suggère un taux de réception de paquets faibles.

### 5.1.4 Étude de la corrélation PRR/puissance d'émission

Il est intéressant d'étudier la corrélation entre puissance d'émission et ratio de paquets reçus. Les résultats de la figure 5.5 ont été obtenus parallèlement à ceux concernant le RSSI de la figure 5.3. Une fois encore on constate une corrélation entre puissance d'émission et PRR.

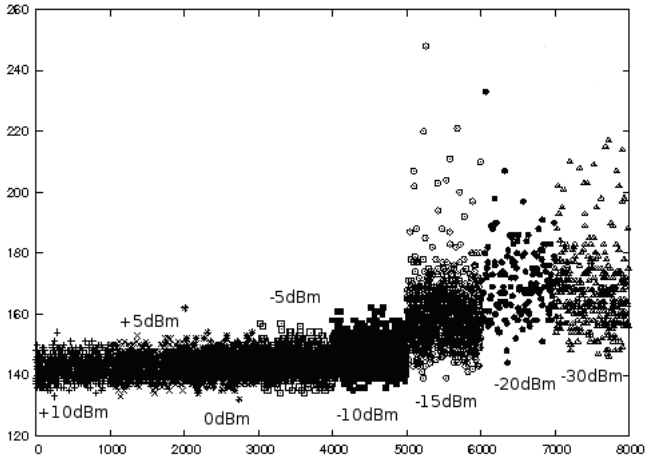



FIGURE 5.4 – *Évolution du LQI avec la puissance de transmission.*

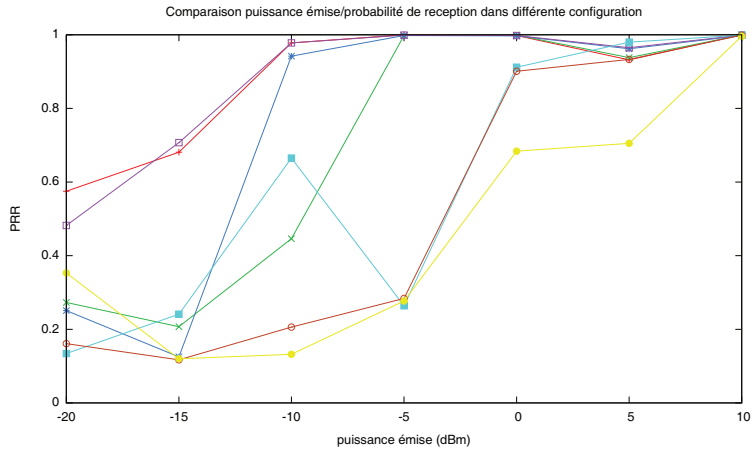



FIGURE 5.5 – *Évolution du ratio de paquets reçus avec l'augmentation de la puissance de transmission.*

On peut également voir un "décrochage" dans les probabilités de réception. Alors qu'elles sont autour de 1, elles chutent assez brusquement lorsque la puissance de transmission diminue. Si l'on rapproche ce graphe de celui de la figure 5.3, on constate que ce décrochage se produit lorsque le RSSI dépasse $-70$. Lors de l'ajustement de puissance de transmission, il est raisonnable de s'assurer que tout voisin conservé le soit tel que le RSSI reste supérieure à cette valeur seuil.
L'étude de la corrélation entre RSSI, LQI et PRR nous apporte les indications suivantes :

- Si un signal est suffisamment fort, il est possible de prévoir son évolution lors de la variation de la puissance de transmission ;
- Il existe des valeurs de LQI discriminantes. Nous les utilisons pour écarter d'emblée certains nœuds du voisinage potentiel d'un autre nœud (mécanisme décrit dans la section 4.4 du chapitre 4) ;
- De la même manière, certaines valeurs du RSSI sont à éviter car elles n'apportent pas un PRR suffisant ou ne permettent pas une bonne prédiction de l'évolution du signal par rapport à l'évolution de la puissance de transmission.

## 5.2 Adaptation du graphe des voisins relatifs (RNG)

Dans cette section, nous décrivons la démarche adoptée pour l'utilisation du RNG dans Goliath.

### 5.2.1 Calcul du graphe RNG

Nous adoptons ici la notation suivante. LQI(u, v) est la valeur du LQI au niveau du nœud $v$ lorsqu'il reçoit un paquet de $u$. De manière similaire, LQI(u, v) est la valeur du RSSI pour un paquet envoyé de $u$ à $v$. On note $P$ l'ensemble ordonné des puissances d'émission disponibles. Dans la littérature, on prend généralement $P = [0; R]$, $R$ étant la portée de communication maximale. La puissance d'émission minimale pour atteindre $v$ depuis $u$ est notée $\mathrm{P}_{\min}(u, v)$.
Le coût d'un lien entre $u$ et $v$ est donné par la fonction de poids $c$ de $E$ dans $A$ où $(A, \prec)$ est un groupe muni d'un opérateur d'ordre $\prec$.
De plus, si $c(u, v) > c(u, w)$, alors la puissance de transmission minimale pour atteindre $v$ depuis $u$ est plus importante que celle nécessaire pour atteindre $w$ depuis $u$.
La fonction de poids la plus couramment utilisée est celle de la distance euclidienne $d$ de $E$ dans $\mathbb{R}^+$. Comme nous supposons que les capteurs n'ont pas la possibilité de connaître la distance qui les sépare, nous utiliserons le RSSI comme fonction de coût. Le calcul du graphe RNG original dans le cadre d'une utilisation avec une fonction de poids est le suivant :

**Input** : $u \in V$, N(u),$N^2(u)$, $k$
**Output** : $\mathrm{N}_{\mathrm{RNG}} \supseteq N(u)$
$N_{RNG}(u) = N(u)$ ;
**foreach** $v \in N(u)$ **do**
  **if** $\exists w \in N_{RNG}(u) \mid (c(u,v) \prec c(u,w)) \wedge (c(u,v) \prec c(w,v))$ **then**
    $N_{RNG}(u) = N_{RNG}(u) \backslash \{v\}$ ;
  **end**
**end**

Algorithme 12 – *Calcul du graphe des voisins relatifs.*

Le calcul du RNG est donc un processus à deux phases :

1. **Découverte de voisinage :** elle est effectuée par l'envoi des messages HELLO dans Goliath,

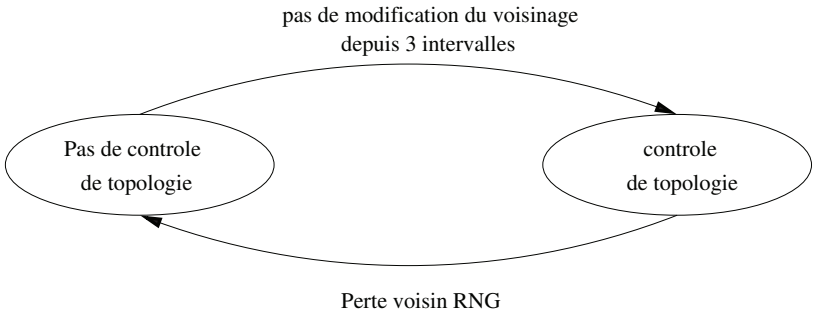



FIGURE 5.6 – *Contrôle de topologie dans* GOLIATH. *Il s'effectue si il n'y a pas eu de changement dans le voisinage du nœud durant 3 intervalles d'envoi de message HELLO. Les changements pris en compte sont la perte d'un lien symétrique, l'apparition d'un nouveau lien symétrique ou la perte de connexion avec le père du nœud.*

2. **Exécution de l'algorithme RNG** : chaque nœud éxécute l'algorithme 12.

Une question délicate est le choix de l'exécution de l'algorithme RNG. Nous choisissons de n'effectuer le contrôle de topologie qu'une fois que la découverte de voisinage semble être complète. Par conséquent, un nœud attendra que son voisinage soit stable. Cette stabilité est vérifiée par le fait qu'il n'y ait pas eu de changement parmi les liens symétriques du nœud durant les 3 derniers intervalles d'envoi de messages HELLO, c'est-à-dire qu'aucun lien symétrique n'ait disparu ou apparu. Nous ne nous intéressons qu'au lien symétrique pour assurer la symétrie du graphe résultant. Ce processus est illustré par la figure 5.6. Un nœud ayant effectué le contrôle de topologie et perdant un lien symétrique sélectionné comme RNG retourne dans l'état sans contrôle de topologie et réutilise la puissance de transmission maximale.

### 5.2.2 Ajustement de la portée de transmission

Nous utilisons les résultats obtenus lors de l'étude de la corrélation entre RSSI, LQI, PRR et puissance de transmission. Si l'on regarde la figure 5.3, on constate que lorsque l'on ne considère que les valeurs de RSSI supérieures à $-70$, on voit qu'elles ont toutes la même pente. Par conséquent, pour approximer le RSSI en fonction d'une puissance de transmission donnée $P_{Tx}$,$P_{Tx} \in P$, nous utilisons la fonction suivante :

$$RSSI(P_{Tx})[dBm] = \alpha.P_{Tx}[dBm] + C_d,$$

avec $C_d$ une constante dépendante de la distance entre l'émetteur et le récepteur et des conditions de l'environnement. Avant que $u$ n'ajuste sa portée de transmission $P^u_{\mathrm{Tx}}$, celle-ci vérifie :

$$RSSI(P^u_{\mathrm{Tx}}) = P_{(u,v_{min})} \quad \Leftrightarrow \quad \alpha.P^u_{\mathrm{Tx}} + C_d = P_{(u,v_{min})}$$

En admettant que l'on veuille conserver un RSSI supérieure ou égale à $rssi\_thres$, la nouvelle portée de transmission de $u$, notée $P^u_{\mathrm{Tx\_RNG}}$, doit vérifier :

$$\begin{aligned} & RSSI(P^u_{\mathrm{Tx\_RNG}}) = rssi\_thres \\ \Leftrightarrow \quad & \alpha.P^u_{\mathrm{Tx\_RNG}} + C_d = rssi\_thres \end{aligned}$$

Des deux équations précédentes on déduit la valeur de la nouvelle puissance de transmission à utiliser :

$$P^u_{\mathrm{Tx\_RNG}} \quad = \quad \frac{rssi\_thres - min}{\alpha} + P^u_{\mathrm{Tx}}.$$

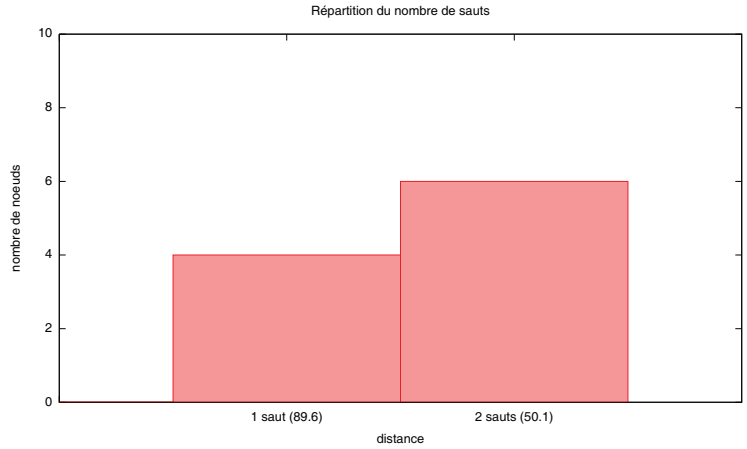



FIGURE 5.7 – *Répartition des distances en nombre de sauts du puits. Les nombres entre parenthèses donnent le pourcentage de temps moyen passé à cette distance en nombre de sauts par les nœuds se trouvent à cette distance la majorité du temps.*

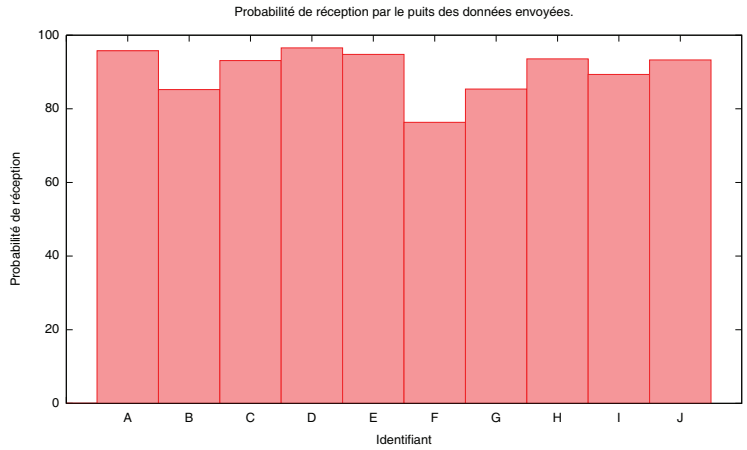



FIGURE 5.8 – *Pourcentage de réception par le puits des messages envoyés par les nœuds.*

On choisira la portée de transmission $P'_{Tx} \in P$ telle que $P'_{Tx} \geq P^{u}_{\text{Tx\_RNG}}$.

## 5.3 ÉVALUATION

Les conditions d'évaluation sont identiques à celle utilisée pour l'évaluation de GOLIATH. Dix nœuds sont déployés à des positions arbitraires. Celles-ci sont cependant identiques à celles utilisées pour l'évaluation de GOLIATH sans contrôle de topologie.

**Distance au puits :** La figure 5.7 donne la distance en nombre de sauts au puits. Quatre nœuds se trouvent majoritairement à 1 saut du puits (en moyenne, 89.6% du temps). Les 6 autres sont 50.1% du temps à 2 sauts du puits. La tendance s'est donc inversée, la majorité des nœuds n'étant plus à 1 saut.

**Pourcentage de paquets reçus :** La figure 4.18 donne, pour chaque nœud, le pourcentage de paquets acheminés jusqu'au puits. On constate une amélioration par rapport à la version sans contrôle de topologie. La majorité des nœuds (excepté le nœud $F$) ont un pourcentage de paquets correctement acheminés supérieur à 80%.

**Répartition de l'utilisation des puissances de transmission :** La figure 5.9 montre la

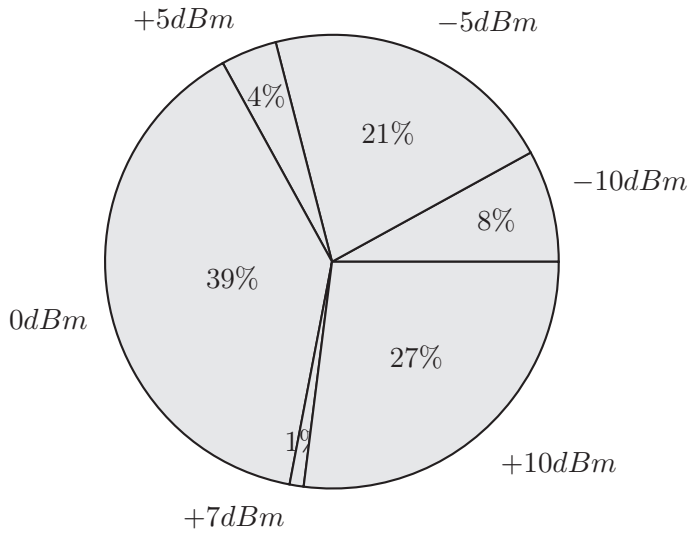



FIGURE 5.9 – *Utilisation moyenne des puissances de transmission pour tous les nœuds. Ainsi, la puissance de transmission de* 0 *dBm est utilisée dans* 39% *des cas.*

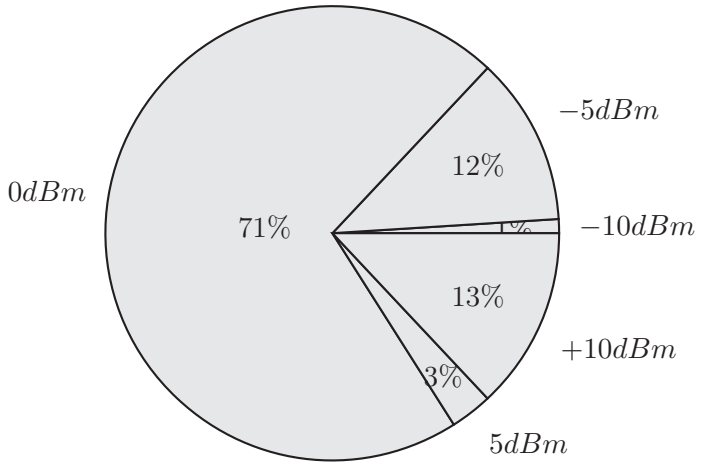



FIGURE 5.10 – *Utilisation moyenne des puissances de transmission pour le nœud J.*

répartition de l'utilisation des puissances de transmission. Dans 39% des cas, la puissance de 0 dBm est utilisée, alors que la puissance de transmission maximale, +10 dBm, est employé dans 27% des cas. On constate donc une baisse assez importante des puissances de transmission utilisée.
**Exemple du nœud** $J$ **:** La figure 5.10 montre l'utilisation des puissances de transmission par le nœud $J$, pris en exemple dans l'évaluation de GOLIATH sans contrôle de topologie. $J$ utilise majoritairement la puissance de 0dBm et dans 12% du temps, il est à une puissance de transmission de −5dBm.

## 5.4 Conclusion du chapitre

Nous avons présenté dans ce chapitre l'utilisation d'un algorithme de réduction dans une implémentation réelle. Pour se faire, nous avons dans un premier temps étudier la corrélation entre RSSI, LQI et PRR. Cette étude nous a permis de déduire des valeurs discriminantes de LQI

servant à écarter un voisin potentiel. Nous avons également mis en évidence la corrélation forte entre RSSI et puissance de transmission, ainsi qu'entre PRR et puissance de transmission pour certaines valeurs du RSSI. Nous avons déduit de cette étude une manière d'adapter la puissance de transmission à nœud donné avec pour contrainte de préserver un RSSI et un PRR suffisant. Nous avons ensuite évalué cette solution avec la pile de communication GOLIATH. Les résultats montrent une amélioration du taux de réception des messages et une meilleure utilisation des puissances de transmission.

# Conclusions et perspectives 6

« But as sure as the sun a shine,
we'll find it in our time. »

**Groundation**,
*Confusing Situation - Young Tree*.

Nous avons étudié la mise en pratique d'algorithmes de contrôle de topologie. Il s'agissait de permettre une utilisation réelle des algorithmes définis dans le cadre de l'utilisation d'un modèle de couche physique à seuil. Notre démarche s'est articulé en deux points. Nous avons d'abord étudié un modèle de couche physique plus réaliste : le modèle probabiliste. Nous y avons défini la notion de $\lambda$-proximité permettant de juger de la qualité d'une topologie. Nous avons ensuite proposé différent algorithme de réduction de graphe adapté au modèle probabiliste et conservant la $\lambda$-proximité. Nous nous sommes ensuite intéressé au problème d'assignement de portée en proposant différentes solutions centralisées au problème. Après avoir démontré que ce problème ne possédait pas de solution localisée, nous avons défini une problématique localisée à laquelle nous avons proposé plusieurs solutions, dont une basée sur une réduction de graphe. Ces solutions permettent de diminuer l'énergie consommée en réduisant la puissance de transmission de chaque nœud.
Nous avons ensuite présenté Goliath, la pile de communication que nous avons implémentée. Goliath met en œuvre différents mécanismes originaux, comme celui du processus de reconnaissance de voisinage et le gestionnaire de buffers. L'évaluation de Goliath a montré qu'elle est capable d'assurer un taux de réception des messages par le puits satisfaisante.
L'étape suivante a été d'intégrer le contrôle de topologie dans Goliath. La phase préliminaire consistait à étudier la corrélation entre l'indicateur de puissance du signal reçu, l'indicateur de qualité du lien, le ratio de paquets correctement reçus et la variation de la puissance de transmission. À partir de cette étude, nous avons sélectionnés le RSSI comme fonction de poids pour le calcul du graphe des voisins relatifs. Nous avons également la méthode utilisée pour l'ajustement de portée effectif à un nœud dont seul le RSSI quand il envoie un paquet est reçu. L'évaluation du contrôle de topologie dans Goliath montre un gain de performance concernant la réception par le puits des messages envoyés par les nœuds du réseau. On constate également que la majorité des nœuds cesse d'utiliser la puissance de transmission maximale.

Parmi les possibles travaux dans la continuité de ceux-ci on peut citer l'évaluation de Goliath en cas de mobilité des nœuds et l'adaptation du contrôle de topologie en conséquence. Il est probable qu'il faille une étude plus poussée du RSSI par le nœud pour opérer l'ajustement de portée. Il s'agirait de déduire de ce RSSI si le nœud s'approche ou au contraire s'éloigne pour savoir s'il est possible de diminuer la puissance de transmission du nœud. Il serait également

intéressant d'évaluer les performances d'une solution proposant un ajustement de portée au niveau du nœud mais également au niveau voisin et paquet. Toujours d'un point de vue expérimental, un sujet intéressant serait de déduire des seuls RSSI et LQI un pourcentage de paquets reçus. Un tel modèle permettrait l'utilisation directe des algorithmes proposés dans le cadre de la $\lambda$-proximité. En dehors du contrôle de topologie, il serait également intéressant de transposer les différentes solutions proposées avec l'utilisation d'un modèle de couche physique à seuil à un modèle de couche physique probabiliste.

# Liste des figures

# Bibliographie


[1] Ieee standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements part 15.4 : Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans). *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pages 1–305, 2006.

[2] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Comput. Commun.*, 30(14-15) :2826–2841, 2007.

[3] H. Badis and K. A. Agha. QOLSR, QoS routing for ad hoc wireless networks using OLSR. *European Transactions on Telecommunications*, 16(5) :427–442, 2005.

[4] G. Barrenetxea, F. Ingelrest, G. Schaefer, and M. Vetterli. The hitchhiker's guide to successful wireless sensor network deployments. In *SenSys*, pages 43–56, 2008.

[5] D. M. Blough, M. Leoncini, G. Resta, and P. Santi. The k-neigh protocol for symmetric topology control in ad hoc networks. In *MobiHoc '03 : Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 141–152, New York, NY, USA, 2003. ACM.

[6] A. Busson, N. Mitton, and E. Fleury. An analysis of the MPR selection in OLSR and consequences. In *Mediterranean Ad Hoc Networking Workshop (MedHocNet'05), Mediterranean Ad Hoc Networking Workshop (MedHocNet'05),*, page 0000, France, 06 2005.

[7] J. Carle and D. Simplot-Ryl. Energy-efficient area monitoring for sensor networks. *Computer*, 37(2) :40–46, 2004.

[8] E. Chávez, N. Mitton, and H. Tejeda. Routing in wireless networks with position trees. In *ADHOC-NOW*, pages 32–45, 2007.

[9] ChipCon. CC1101 RF transceiver data sheet.

[10] T. Clausen and P. Jacquet. Optimized link state routing protocol (olsr). RFC 3626, Internet Engineering Task Force, Oct. 2003.

[11] A. E. F. Clementi, P. Penna, and R. Silvestri. On the power assignment problem in radio networks. *Mob. Netw. Appl.*, 9(2) :125–140, 2004.

[12] E. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9), September 1965.

[13] L. Ding and Z.-H. Guan. Modeling wireless sensor networks using random graph theory. *Physica A : Statistical Mechanics and its Applications*, 387(12) :3008 – 3016, 2008.

[14] A. Dunkels. Full TCP/IP for 8 bit architectures. In *Proceedings of First ACM/Usenix International Conference on Mobile Systems, Applications and Services (MobiSys 2003)*, page 14, San Francisco, USA, 2003.

[15] A. Dunkels, B. Grönvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004.

[16] A. Dunkels, F. Österlind, and Z. He. An adaptive communication architecture for wireless sensor networks. In *SenSys '07 : Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 335–349, New York, NY, USA, 2007. ACM.

[17] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads : Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems (SenSys 2006)*, Boulder, Colorado, USA, Nov. 2006.

[18] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5 :17–61, 1960.

[19] L. M. Feeney. An energy-consumption model for performance analysis of routing protocols for mobile ad hoc networks. *ACM J. of Mobile Networks and Applications*, 3 :239–249, 2001.

[20] FreeRTOS a Free RTOS for Embedded Systems. http ://www.freertos.org.

[21] K. R. Gabriel and R. R. Sokal. A New Statistical Approach to Geographic Variation Analysis. *Syst Biol*, 18(3) :259–278, 1969.

[22] A. Gallais, J. Carle, D. Simplot-Ryl, and I. Stojmenovic. Localized sensor area coverage with low communication overhead. *Mobile Computing, IEEE Transactions on*, 7(5) :661–672, May 2008.

[23] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesc language : A holistic approach to networked embedded systems. In *PLDI '03 : Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, pages 1–11, New York, NY, USA, 2003. ACM.

[24] C. Gomez, A. Cuevas, and J. Paradells. Ahr : a two-state adaptive mechanism for link connectivity maintenance in aodv. In *REALMAN '06 : Proceedings of the 2nd international workshop on Multi-hop ad hoc networks : from theory to reality*, pages 98–100, New York, NY, USA, 2006. ACM.

[25] T. W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs (Pure and Applied Mathematics (Marcel Dekker))*. CRC, January 1998.

[26] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *HICSS '00 : Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8*, page 8020, Washington, DC, USA, 2000. IEEE Computer Society.

[27] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ASPLOS-IX : Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104, New York, NY, USA, 2000. ACM.

[28] F. Ingelrest, N. Mitton, and D. Simplot-Ryl. A turnover based adaptive hello protocol for mobile ad hoc and sensor networks. In *Proc. 15th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2007)*, 2007.

[29] F. Khadar and D. Simplot-Ryl. Connectivity and topology control in wireless ad hoc networks with realistic physical layer. In *ICWMC '07 : Proceedings of the Third International Conference on Wireless and Mobile Communications*, page 49, Washington, DC, USA, 2007. IEEE Computer Society.

[30] L. M. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power consumption in packet radio networks. volume 243, pages 289–305, Essex, UK, 2000. Elsevier Science Publishers Ltd.

[31] J. Kuruvila, A. Nayak, and I. Stojmenovic. Hop count optimal position-based packet routing algorithms for ad hoc wireless networks with a realistic physical layer. *Selected Areas in Communications, IEEE Journal on*, 23(6) :1267–1275, June 2005.

[32] N. Li, J. Hou, and L. Sha. Design and analysis of an mst-based topology control algorithm. *Wireless Communications, IEEE Transactions on*, 4(3) :1195–1206, May 2005.

[33] S. Lin, J. Zhang, G. Zhou, L. Gu, J. A. Stankovic, and T. He. ATPC : adaptive transmission power control for wireless sensor networks. In *SenSys '06 : Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 223–236, New York, NY, USA, 2006. ACM.

[34] L. Moraru and D. Simplot-Ryl. Qos preserving topology advertising reduction for olsr routing protocol for mobile ad hoc networks. In *WONS 2006 : Third Annual Conference on Wireless On-demand Network Systems and Services*, 2006.

[35] W. Peng and X.-C. Lu. On the reduction of broadcast redundancy in mobile ad hoc networks. In *MobiHoc '00 : Proceedings of the 1st ACM international symposium on Mobile ad hoc networking & computing*, pages 129–130, Piscataway, NJ, USA, 2000. IEEE Press.

[36] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. *Mobile Computing Systems and Applications, IEEE Workshop on*, 0 :90, 1999.

[37] R. C. Prim. Shortest connection networks and some generalizations. *Bell Systems Technical Journal*, pages 1389–1401, Nov. 1957.

[38] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Comput. Surv.*, 37(2) :164–194, 2005.

[39] I. Stojmenovic and X. Lin. Power-aware localized routing in wireless networks. *Parallel and Distributed Systems, IEEE Transactions on*, 12(11) :1122–1133, Nov 2001.

[40] I. Stojmenovic, A. Nayak, J. Kuruvila, F. J. Ovalle-Martínez, and E. Villanueva-Pena. Physical layer impact on the design and performance of routing and broadcasting protocols in ad hoc and sensor networks. *Computer Communications*, 28(10) :1138–1151, 2005.

[41] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *Parallel and Distributed Systems, IEEE Transactions on*, 13(1) :14–25, Jan 2002.

[42] TI. TI FAQ : Calculation and usage of LQI and RSSI. http ://e2e.ti.com/forums/p/362/29797.aspx.

[43] D. Tian and N. D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. In *WSNA '02 : Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41, New York, NY, USA, 2002. ACM.

[44] G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12 :261–268, 1980.

[45] A. Troël, F. Weis, and M. Banâtre. Découverte automatique entre terminaux mobiles communicants. Research Report RR-4979, INRIA, 2003.

[46] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust : Communicating with a cubic-millimeter computer. *Computer*, 34(1) :44–51, 2001.

[47] R. Wattenhofer, L. Li, P. Bahl, and Y. min Wang. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. pages 1388–1397, 2001.

[48] R. Wattenhofer and A. Zollinger. Xtc : a practical topology control algorithm for ad-hoc networks. *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 216–, April 2004.

[49] J. Wieselthier, G. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2 :585–594 vol.2, 2000.

[50] J. Wu and H. Li. A dominating-set-based routing scheme in ad hoc wireless networks. *Telecommunication Systems Journal*, 3 :63–84, 1999.

[51] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed energy conservation for ad hoc routing. In *MobiCom '01 : Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 70–84, New York, NY, USA, 2001. ACM.

[52] F. Xue and P. R. Kumar. The number of neighbors needed for connectivity of wireless networks. *Wirel. Netw.*, 10(2) :169–181, 2004.

[53] G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic. Models and solutions for radio irregularity in wireless sensor networks. *ACM Trans. Sen. Netw.*, 2(2) :221–262, 2006.

**Titre** Contrôle de topologie dans les réseaux de capteurs : de la théorie à la pratique.

**Résumé** Les réseaux de capteurs sont des réseaux composés de petits objets communicants à faibles ressources incluant capacité de calcul et batterie. Ils sont notamment conçus pour être déployés dans des zones à risque ou difficile d'accès. Chaque capteur recueille des informations sur son environnement qu'il envoie, à l'aide d'une interface radio, à une entité responsable du traitement de ces données en utilisant les autres capteurs du réseau comme relais si nécessaire. Lors d'un tel déploiement, un nombre important de capteurs sont mis en place pour s'assurer de la couverture de la zone à surveiller. Ce nombre important implique que chaque capteur maintienne, pour chacun des capteurs avoisinants, des informations telles que leur activité. Le maintien de telles informations est coûteux tant en terme de communication et de mémoire. L'objectif de cette thèse est de permettre de réduire le nombre de ces capteurs avoisinants tout en préservant les fonctionnalités du réseau. L'ensemble des capteurs avoisinants est défini par la portée de communication radio. Le contrôle de topologie vise à réduire l'ensemble des voisins logiques d'un nœud, tandis que le contrôle de portée vise à sélectionner la meilleure portée pour atteindre ces voisins tout en réduisant l'ensemble physique des voisins d'un nœud. Ces travaux se proposent d'étudier, d'un point de vue théorique, les performances pouvant être obtenue par le contrôle de topologie et le contrôle de puissance. Nous étudions des algorithmes définis dans le cadre de l'utilisation d'un modèle de couche physique idéale, et les adaptons à une utilisation dans le cadre de l'utilisation d'un modèle de couche physique réaliste. D'un point de vue pratique, nous avons développé une pile de communication, Goliath, intégrant le contrôle de topologie et de puissance. Nous évaluons les performances de Goliath avant et sans contrôle de topologie. Lors de l'utilisation du contrôle de puissance dans Goliath, les capteurs utilisent une puissance de transmission inférieure à la puissance maximale sans impacter les performances du réseau.

**Mots-clés** réseaux de capteurs, contrôle de topologie, contrôle de puissance, pile de communication.

**Title** Topology control in Wireless Sensor Networks : from Theory to Practice.

**Abstract** Wireless Sensor Networks are networks of small communicating devices with constrainted resources. They are usually deployed in risky or difficult to access areas. Each sensor gathers information about its environment and send it to a dedicated entity using other sensors as relays. A large number of sensors may be deployed to ensure the coverage of given area. This large number implies that each sensor has to maintain, for each neighboring sensor, information about its activity for instance. Maintaining this information is memory consuming and implies a huge communication overhead. The aim of this thesis is to reduce the number of neighboring sensor while keeping network services up. The set of neighboring nodes is defined by the communication range. Topology control aims at logically reducing the number of neighboring sensors while power control aims at physically reducing this number. In this work, we theoretically study the bounds that can be obtained by using topology and power control. We study algorithms defined with a ideal physical layer model and show how to adapt them to be used with a realistic physical layer. On a practical side, we developped a communication stack, Goliath, that includes topology and power control. We evaluate the performances of Goliath with and without topology control and power adjustment. When Goliath uses power adjustment, sensors use a transmission power smaller than the maximum power without any impact of the performances of the network.

**Keywords** wireless sensor networks, topology control, power control, communication stack.