# Les arbres couvrants de la théorie à la pratique. Algorithmes auto-stabilisants et réseaux de capteurs

Fadwa Boubekeur

**UNIVERSITÉ PIERRE ET MARIE CURIE**
**UPMC SORBONNE UNIVERSITÉS**

École doctorale

**Informatique, Télécommunications et Électronique (Paris)**

réalisée au

**Laboratoire Informatique de Paris 6**

# Thèse de Doctorat

Pour obtenir le grade de

## Docteur de l'Université Pierre et Marie Curie

Présentée par

**Fadwa Boubekeur**

Sujet de thèse

# Les arbres couvrants de la théorie à la pratique
## Algorithmes auto-stabilisants et Réseaux de Capteurs

à soutenir le 12 Octobre 2016 devant le jury composé de :

| | |
|---|---|
| **Pr. Andrzej Duda** | **Rapporteur** |
| **Pr. Colette Johnen** | **Rapportrice** |
| **Dr. Marcelo Dias di Amorin** | **Examinateur** |
| **Dr. Jeremie Leguay** | **Examinateur** |
| **Pr. Franck Petit** | **Examinateur** |
| **Pr. Vincent Villain** | **Examinateur** |
| **Dr. Lélia Blin** | **Directrice de thèse** |

# Table des matières


**1 Introduction . . . . . 7**

1.1 Contexte . . . . . 7

1.2 Contributions de la thèse . . . . . 9

1.3 Organisation du document . . . . . 11

**2 Arbres couvrants auto-stabilisants . . . . . 13**

2.1 Introduction . . . . . 13

2.2 Système distribué . . . . . 14

2.2.1 Pannes dans les systèmes distribués . . . . . 14

2.3 Auto-stabilisation . . . . . 16

2.3.1 Définitions . . . . . 16

2.3.2 Modèle . . . . . 17

2.4 Arbres couvrants sous contraintes . . . . . 20

2.4.1 Notions sur les arbres couvrants . . . . . 20

2.4.2 Algorithmes auto-stabilisants de construction d'arbres couvrants sous-contraintes . . . . . 22

2.5 Conclusion . . . . . 28

**3 Arbre couvrant de diamètre minimum . . . . . 29**

3.1 Introduction . . . . . 29

3.2 Problème de l'arbre couvrant de diamètre minimum . . . . . 30

3.2.1 Etat de l'art . . . . . 30

3.3 Algorithme auto-stabilisant pour le MDiamST . . . . . 31

3.3.1 Modèle . . . . . 31

3.3.2 Vue d'ensemble de l'algorithme . . . . . 32

3.4 Description détaillée de l'algorithme . . . . . 34

3.4.1 Election d'une racine élue et construction d'un arbre couvrant . . . 34

3.4.2 Circulation du jeton . . . . . 34

3.4.3 Calcul de l'excentricité . . . . . 38

3.4.4 Calcul des centres . . . 46
3.4.5 Ordre de priorité entre les règles de notre algorithme . . . 47
3.5 Preuve de l'algorithme . . . 47
3.6 Conclusion . . . 64

**4 Réseaux à faible puissance et à perte . . . 65**
4.1 Introduction . . . 65
4.2 Réseaux de capteurs . . . 66
4.2.1 Domaines d'application . . . 66
4.2.2 Caractéristiques . . . 67
4.2.3 Architecture et couches protocolaires dédiées aux réseaux de capteurs 68
4.3 La technologie 802.15.4 . . . 69
4.3.1 Dispositifs 802.15.4 . . . 69
4.3.2 Spécifications de la IEEE 802.15.4 . . . 70
4.3.3 Modes de communication de la IEEE 802.15.4 . . . 70
4.4 Protocole RPL . . . 71
4.4.1 Trafic supporté . . . 71
4.4.2 La fonction objective . . . 71
4.4.3 Métriques et contraintes de routage . . . 72
4.4.4 Les messages de contrôle . . . 73
4.4.5 DODAG . . . 75
4.4.6 Le trickle timer . . . 78
4.5 Evaluation des performances du protocole RPL . . . 78
4.5.1 Autres protocoles de routage . . . 79
4.6 Améliorations du protocole RPL . . . 80
4.7 Simulations et expérimentations . . . 81
4.8 Conclusion . . . 82

**5 Protocole RPL à degré borné . . . 83**
5.1 Introduction . . . 83
5.2 Problématique . . . 84
5.2.1 Impact de l'instabilité de la topologie d'un réseau de capteurs . . . 84
5.2.2 Impact d'un DODAG non borné . . . 85
5.2.3 Améliorations de RPL . . . 86
5.3 BD-RPL . . . 86
5.3.1 Impact de BD-RPL . . . 87
5.3.2 Mise à jour des routes descendantes . . . 88

5.4 Implémentation de BD-RPL . . . . . 91
5.4.1 Expérience personnelle . . . . . 91
5.4.2 Implémentation . . . . . 92
5.4.3 Configuration . . . . . 93
5.4.4 Simulations . . . . . 93
5.4.5 Expérimentations . . . . . 96
5.5 Conclusion . . . . . 100

**6 Conclusion . . . . . 101**
6.1 Résumé des contributions de la thèse . . . . . 102
6.2 Perspectives . . . . . 102

**Bibliographie 104**
Bibliographie . . . . . 104

**Table des figures . . . . . 111**

# Chapitre 1

# Introduction

Les réseaux informatiques ont considérablement évolué depuis leur apparition dans les années soixante. Il y a eu une grande mutation depuis les réseaux primitifs (composés d'un ordinateur central et de terminaux) en passant par ARPANET (Advanced Research Projects Agency Network) car nous parlons maintenant de réseaux avec des millions d'objects connectés. L'avancée des réseaux informatiques a rendu possible la communication entre toutes sortes d'équipements tels que les ordinateurs, les tablettes, les téléphones portables, les robots ainsi que tout dispositif doté d'une antenne radio et d'un micro-contrôleur. Ces dispositifs peuvent être embarqués sur différents objets (capteurs, robots, machines à café, animaux ou tout objet du quotidien) et être connecté grâce à Internet, nous parlons alors d'Internet des objects.

L'omniprésence programmée de ces objets dans notre quotidien implique la fabrication de dispositifs bon marché à faible puissance. L'internet des objets est donc composé en partie d'objets avec des contraintes en mémoire, en énergie, en puissance de calcul et de transmission. Ces objets (fixes ou mobiles) seront regroupés en des réseaux de plus petite taille appelés réseaux LLN (en anglais *Low Power and Lossy Networks* notés *LLN*). Les réseaux LLN peuvent être constitués de nœuds capteurs capables de faire des mesures (telles que le relevé de température ou de la pollution) et de les transmettre à un nœud puits, on parle alors de réseaux de capteurs sans fils ( *WSN*).

## 1.1 Contexte

Les réseaux de capteurs sont utilisés dans plusieurs domaines, par exemple, l'environnement, l'industrie, les applications militaires, la santé, la domotique, etc. Les capteurs constituant ces réseaux font des mesures dans l'environnement et transmettent sur des liens radio les informations collectées à un puits ou plusieurs. Le puits agit comme une passerelle entre ce réseau et Internet. Les capteurs communiquent entre eux grâce à des liens radio. La Figure 1.1 représente un réseau de capteurs connecté à Internet.

Dans de tels réseaux, nous avons à faire à des objets qui possèdent peu de ressources : une faible puissance de calcul, une faible puissance de transmission, une faible bande passante, une mémoire de stockage limitée ainsi qu'une batterie la plus part du temps à durée de vie elle aussi limitée. De telles caractéristiques ajoutées aux obstacles physiques, aux interférences et à la variation de la qualité des liens radio font que les réseaux de capteurs sont des réseaux non fiables.

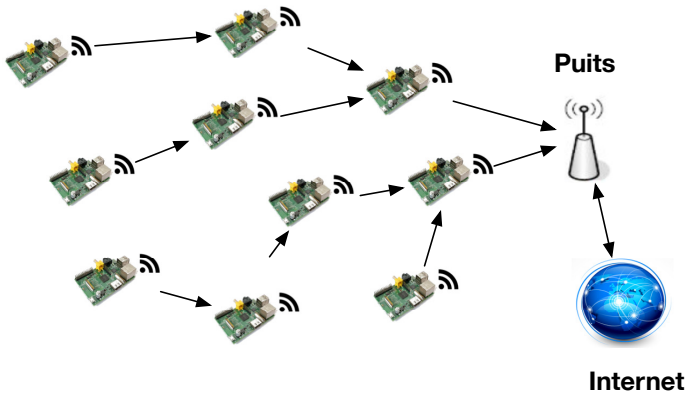



FIGURE 1.1 – Réseau de capteurs

L'IEEE et l'IETF ont fait de grands efforts pour l'élaboration de protocoles qui prennent en charge les contraintes des réseaux de capteurs. L'IEEE a proposé un standard pour la couche physique et MAC appelé *IEEE 802.15.4*. Ce standard répond à une large gamme de scénarios d'applications. D'autre part, l'IETF comprend plusieurs groupes concernant les réseaux de capteurs, deux nous intéressent particulièrement. Le premier groupe *6LoWPAN* a développé le protocole 6LoWPAN pour permettre la connectivité IPv6 de bout en bout dans les réseaux de capteurs. Le deuxième groupe appelé *ROLL* pour Routing Over Low-power Lossy-links a proposé un protocole de routage qui prend en charge les exigences des réseaux de capteurs au niveau réseau appelé *RPL*. RPL (pour Routing Protocol for Low Power and Lossy Networks) est un protocole de routage dédié aux réseaux multi-sauts qui construit de façon proactive une topologie logique appelée *DODAG* (Destination Oriented Directed Acyclic Graph) pour l'acheminement des données collectées et ceci pour différents types de trafic : point à point, point à multi-points et multi-points à point.

Le but de l'algorithmique distribuée est de concevoir des protocoles pour les réseaux, à l'inverse de l'algorithmique dite "centralisée", l'algorithmique distribuée nécessite l'échange d'informations et la coopération entre les entités afin de résoudre un but global. Les réseaux étant de plus en plus grands et déployés dans des environnements hostiles, il serait irréaliste de considérer l'absence totale de pannes. Ces pannes peuvent être la perte de liens de communication, la disparition de nœuds, des attaques malveillantes, etc. Il faut donc concevoir des d'algorithmes distribués tolérants aux pannes. L'auto-stabilisation est une branche de l'algorithmique distribuée qui assure qu'à la suite d'une ou de plusieurs pannes transitoires, le système va retrouver de lui-même un comportement correcte au bout d'un temps fini.

Dans les réseaux de communication distribués à large échelle, il est difficile de prévoir les fautes qui vont se produire. Garantir la disponibilité et la fiabilité de ces réseaux est crucial. La structure de communication mise en place entre les entités de calcul doit être tolérante aux fautes et donc auto-stabilisante.

Les échanges d'informations doivent être optimisés afin d'éviter la surcharge des liens de communication qui pourrait entrainer une paralysie du réseau. Une structure de communication adéquate pour répondre à ce type de problème est l'arbre couvrant le réseau [Per85]. En effet, un arbre couvrant est une structure de communication qui maintient un unique

chemin entre toutes paires de nœuds, tout en minimisant le nombre de liens de communications utilisés. Ces deux propriétés font de l'arbre couvrant la structure de communication la plus utilisée dans les réseaux. Il n'existe pas un unique arbre couvrant pour un réseau donné mais une multitude d'arbres couvrants suivant la qualité de communication souhaitée. L'arbre couvrant construit dépend du facteur que l'on souhaite optimiser, nous appelons ces arbres couvrants des arbres couvrants sous contraintes. Par exemple, si l'on souhaite minimiser la communication d'un nœud particulier vers les autres nœuds du réseau, on construira un arbre couvrant en largeur d'abord (*BFS*). Si l'on souhaite minimiser le délai de communication entre tous les nœuds, on construira un arbre couvrant de diamètre minimum. Il existe bien d'autres arbres couvrants sous contraintes. La construction d'arbres couvrants sous contraintes a été largement étudiée dans la littérature, nous donnerons un état de l'art non exhaustif dans cette thèse.

Nous considérons dans cette thèse l'aspect acheminement de données sur le DODAG. En effet, dans un DODAG RPL, un nœud possède plusieurs parents mais lors de la transmission de données, il n'utilise qu'un seul parent. L'acheminement des données se fait donc de saut en saut d'un enfant à son parent, c'est pour cela que nous abordons le DODAG construit par le protocole RPL comme un arbre couvrant enraciné à une racine (nœud puits). Optimiser la construction du DODAG RPL revient donc à optimiser la construction d'un arbre couvrant selon une contrainte donnée.

De plus, nous considérons les contraintes des réseaux de capteurs en termes de haute dynamicité (batterie déchargée, variabilité du lien radio) comme des fautes transitoires. Ceci nous conduit par conséquent à construire une structure couvrante tolérante aux fautes transitoires. L'auto-stabilisation est une réponse à ce problème car elle se traduit par la capacité du réseau à se reconstruire en présence de fautes transitoires (qui sont la haute variabilité des liens radio dans un réseau de capteurs). Concevoir des réseaux de capteurs robustes et auto-stabilisants est une question ouverte et fondamentale dans la recherche et l'industrie. C'est pourquoi, la construction d'arbres couvrants auto-stabilisants est au centre de cette thèse.

## 1.2 Contributions de la thèse

Le but de la thèse est de proposer des algorithmes auto-stabilisants dédiés aux réseaux de capteurs. Cette thèse a donc donné lieu à deux contributions principales.

#### Arbre couvrant de diamètre minimum.

Dans un réseau de capteurs sans fils, l'organisation de la communication se fait autour d'un nœud appelé *racine*. Lorsque nous souhaitons minimiser le délai de communication entre tous les nœuds, choisir quel nœud deviendra racine devient l'enjeux principal. En effet, minimiser le diamètre de l'arbre couvrant revient à construire un BFS enraciné en un centre du graphe. La Figure 2.4.2 donne un exemple d'un arbre couvrant de diamètre minimum, l'entier présent dans le nœud représente la distance à la racine.

Une contrainte importante dans les réseaux de capteurs est l'espace mémoire disponible pour chaque nœud. C'est pour cela que la principale mesure de complexité que nous considérons est la mémoire utilisée par chaque nœud. Dans ce cadre, la première contribution de la thèse est un algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum. Notre algorithme possède plusieurs avantages. Le premier,

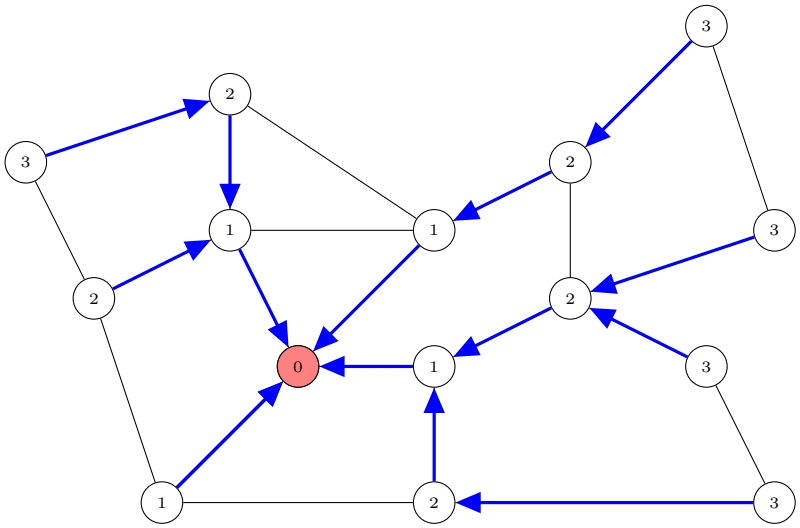



FIGURE 1.2 – Exemple d'arbre couvrant de diamètre minimum.

est l'occupation mémoire requise par chaque nœud. Le seul algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum nécessite $O(n \log n)$ bits de mémoire par nœud [BLB95]. Alors que notre algorithme utilise une mémoire de $O(\log n)$ bits par nœud, ce qui réduit le résultat précédent d'un facteur $n$. Notre algorithme conserve un temps de convergence polynomial. Le deuxième avantage de notre algorithme, est qu'il fonctionne sous un environnement réaliste dans un réseau de capteurs car nous considérons un système totalement asynchrone. Notre algorithme est le premier algorithme pour ce problème qui fonctionne sous un démon distribué non équitable. En d'autres termes, aucune restriction n'est faite sur le comportement asynchrone du réseau. Ce travail a donné lieu à des publications dans une conférence internationale [BBD15] et nationale [BBD16], et une version journal est actuellement en soumission.

**Protocole RPL à degré borné.**

Nous avons étudié dans cette thèse le fonctionnement du protocole RPL et plus particulièrement la topologie (*DODAG*) qu'il construit. Les évaluations de performances du protocole RPL ont conclu que le protocole construisait une topologie DODAG instable [ITN13]. La plupart des solutions préconisent de définir de nouvelles métriques de routage plus adaptées aux variations des liens radios lors de la construction du DODAG. Ces métriques sont généralement basées sur une certaine évaluation des liens radio. Contrairement aux solutions précédentes, nous avons adopté une nouvelle approche pour résoudre l'instabilité des routes du DODAG construit par RPL. Nous nous sommes intéressés à la construction du DODAG en lui-même, en plaçant une contrainte additionnelle sur le nombre d'enfants qu'un nœud peut accepter durant sa construction. Cette contrainte a pour effet de réduire le taux de changement de parent et par conséquent améliorer les performances du protocole en termes de taux de délivrance de paquets, de délai et de consommation énergétique. La Figure 1.2 donne un exemple d'un arbre couvrant de degré borné, L'entier présent dans le nœud représente l'identifiant du nœud.

Nous avons constaté un défaut d'implémentation de RPL qui réside dans la non mise à jour des routes descendantes. Nous avons par conséquent proposé un mécanisme de mise

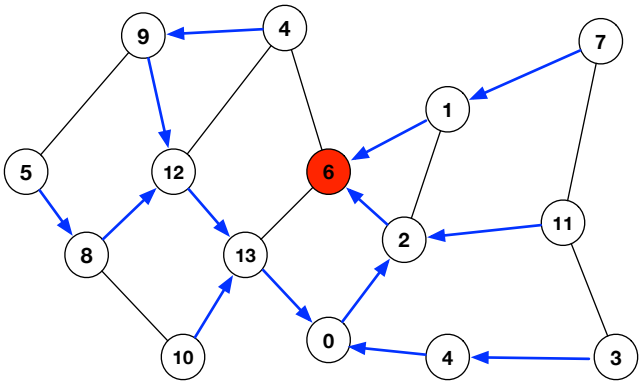



FIGURE 1.3 – Exemple d'arbre couvrant de degré borné.

à jour de ces routes. Notre solution ne génère pas un surplus de messages de contrôle car nous utilisons les messages de contrôle existants fournis par RPL pour borner le DODAG et le mettre à jour. D'autre part, notre protocole appelé *BD-RPL* ne dépend pas de la métrique de routage utilisée. Autrement dit, toute amélioration des métriques de routage, pourra être utilisé par BD-RPL pour obtenir un protocole encore plus performant. Enfin, cette contribution est double puisque nous avons validé nos résultats à la fois en simulations et en expérimentations. Les expérimentations ont prouvé une amélioration par rapport à RPL d'une moyenne de 10% dans le taux de délivrance de paquets, de 50% dans la consommation énergétique et de 60% en délai de transmission. Ce travail a donné lieu à une publication dans un Workshop (Q2SWinet) d'une conférence internationale (MSWiM 2015) [BBLM15] et d'un poster dans une autre conférence internationale (MobiCom 2015) [BBK+15].

## 1.3 Organisation du document

Le document est organisé en 5 chapitres. Le premier chapitre est consacré au paradigme de l'auto-stabilisation et aux algorithmes auto-stabilisants de constructions d'arbres couvrants sous-contraintes. Dans ce chapitre, nous présenterons les différents modèles considérés par les algorithmes auto-stabilisants, les notions liées aux arbres couvrants ainsi qu'un état de l'art non exhaustif sur les algorithmes auto-stabilisants pour la construction d'arbres couvrants sous-contraintes.

Le chapitre suivant présente la première contribution de la thèse, à savoir, un algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum sous un démon distribué non équitable. Dans ce chapitre, nous ferons un état de l'art exhaustif concernant les algorithmes de construction d'arbres couvrants de diamètre minimum, nous décrirons ensuite le modèle que nous utilisons. Nous donnerons ensuite les grands principes de notre algorithme avant d'en faire une description détaillée. Enfin, la dernière partie du chapitre sera consacrée aux preuves liées à notre algorithme, aux preuves de corrections et aux preuves concernant la complexité spatiale et temporelle.

Dans le chapitre suivant, nous abordons le deuxième grand domaine de notre thèse à savoir les réseaux de capteurs. Nous ferons d'abord une introduction sur les réseaux de capteurs, nous présenterons ensuite les technologies utilisées dans cette thèse. Nous

détaillerons notamment le fonctionnement du protocole RPL, principal sujet de ce travail, et ferons un état de l'art. Nous présenterons ensuite les outils de simulations et d'expérimentations utilisés pour valider nos résultats.

Ces résultats ainsi que la solution qui nous y ont menée seront présentés dans le chapitre quatre. Nous décrirons dans ce chapitre notre deuxième contribution à savoir un protocole de construction de DODAG borné. Nous poserons tout d'abord la problématique, nous expliciterons par la suite la solution proposée. Nous présenterons enfin nos résultats en simulations et en expérimentations.

Le dernier chapitre conclut ce document en présentant un résumé des contributions de la thèse ainsi que de futures perpectives que ces travaux ont ouverts.

# Chapitre 2

# Arbres couvrants auto-stabilisants

## 2.1 Introduction

Les systèmes distribués modélisent les réseaux informatiques, ils permettent la coordination à grande échelle entre les entités de calcul d'un réseau. Plus précisément, un système distribué est un ensemble d'entités de calcul interconnectées entre-elles, ces entités de calculs peuvent être représentées par des ordinateurs, des téléphones portables, des robots, des capteurs, ainsi que tout autre objet communicant pouvant être connecté par un réseau. Pour donner à ces entités un terme plus générale et englobant, nous les appellerons par la suite les *nœuds* du réseau. Chaque nœud a la capacité de communiquer qu'avec un sous ensemble de nœuds, sous-ensemble appelé ses voisins. Nous utiliserons la modélisation classique des réseaux à savoir les graphes ; un réseau est modélisé par un graphe où les sommets sont les nœuds et les arêtes les liens de communication.

L'omniprésence des réseaux informatiques dans notre quotidien, la diversité des équipements réseau connectés et l'importance des quantités de données à traiter ont posé de nouveaux défis à la recherche. Des protocoles de routage doivent être conçus en prenant en compte ces évolutions, en particulier en présence de fautes. Une approche classique pour résoudre ce problème est de construire des protocoles de routage distribués auto-stabilisants. L'auto-stabilisation [Dij74] est l'une des techniques les plus polyvalentes pour maintenir la disponibilité des applications réseaux distribuées, leur fiabilité et leur maintenance. Après l'apparition d'une défaillance qui a placé les composants du réseau dans un état global arbitraire, l'auto-stabilisation garantit la récupération vers un comportement légitime en un temps fini sans aucune intervention extérieure. Cette approche est particulièrement bien adaptée pour les réseaux auto-organisés ou autonomes distribués tels que les réseaux de capteurs.

Dans les réseaux, une tâche essentielle est de maintenir des communications efficaces. Une façon naturelle pour faire face à ce problème consiste à construire un arbre couvrant le réseau et d'acheminer les messages entre les nœuds grâce à cette structure. Cette dernière a l'avantage de fournir une et une seule route entre n'importe quelle paire de nœuds du réseau, ce qui permet de faire du routage facilement. Dans un réseau, en fonction des contraintes environnementales, l'arbre couvrant le réseau peut optimiser des métrique différentes. Si l'on souhaite par exemple minimiser le coût d'une structure, l'arbre couvrant construit sera un arbre couvrant de poids minimum. Si, l'on souhaite minimiser la distance vers un nœud particulier, l'arbre couvrant construit sera un arbre couvrant en largeur d'abord. Il existe ainsi, autant de protocoles de construction d'arbres couvrants sous-

contraintes que d'arbres couvrants optimisant une certaine métrique.

Ce chapitre pose le cadre théorique de la thèse. Avant de présenter nos principales contributions, nous allons définir dans ce chapitre plusieurs concepts liés aux arbres couvrants auto-stabilisants.

**Organisation du chapitre** Nous allons tout d'abord rappeler les grandes lignes des systèmes auto-stabilisants dans la section 2.2 et 2.3. Dans la section 2.4, nous allons décrire les notions élémentaires de la théorie des graphes utilisées dans la thèse. Ensuite, nous décrirons les types d'arbres couvrants existants dans la littérature et dresserons un état de l'art non exhaustif sur les algorithmes auto-stabilisants pour la construction d'arbres couvrants sous contraintes. Enfin, dans la section 2.5 nous ferons une conclusion sur les concepts abordés et introduirons notre contribution principale dans le domaine.

## 2.2 Système distribué

Un système distribué est un ensemble d'entités de calcul avec des capacités de communication. Ces entités de calcul, appelées *processeurs*, ou *nœuds*, communiquent les unes avec les autres afin de résoudre une tâche globale donnée. Cette définition est générale car elle englobe plusieurs systèmes tels que les réseaux informatiques, les réseaux téléphoniques, les systèmes parallèles, les systèmes point à point, les réseaux de robots ou les réseaux de capteurs. Ces systèmes ont une propriété principale : *l*a localité, cette localité réside dans le fait qu'aucun nœud n'a de connaissance globale sur le système (sauf précision contraire), chaque nœud a seulement accès à son information locale. Si un nœud a besoin d'informations complémentaires, il peut communiquer uniquement avec les nœuds avec qui il partage un lien de communication.

Dans cette thèse, nous considérons des systèmes distribués tolérants aux fautes transitoires. Nous allons voir les types de pannes dans les systèmes distribués et une réponse à la présence de fautes transitoires dans de tels systèmes : l'auto-stabilisation.

### 2.2.1 Pannes dans les systèmes distribués

Dans un système distribué, tout comportement anormal d'un nœud est qualifié de panne. Une panne, appelée aussi faute, est une défaillance temporaire ou définitive d'un ou de plusieurs composants (nœud ou lien de communication) d'un système. Une faute peut survenir suite à une panne matérielle de l'équipement réseau, à une défaillance dans le logiciel, à une erreur de l'utilisateur ou à un problème du réseau. Plus précisément, dans un réseau de capteurs à large échelle, le nombre conséquent de dispositifs communicants dans le réseau augmente la probabilité de fautes dans le système. Une faute peut survenir suite à la panne d'un nœud capteur pour cause de batterie déchargée ou de défaillance de l'équipement, à une erreur de calcul pour un nœud capteur ou à des perturbations environnementales sur les liens radios. Il existe plusieurs types de fautes qui peuvent survenir dans un système distribué. Dans la littérature, les fautes sont classifiées selon plusieurs critères. Le premier critère prend en compte la *nature* de la faute (franche, byzantine ou sur état) et le deuxième considère la *durée* de la faute (transitoire, intermittente ou permanente).

**Durée de la faute**

— Une faute est *permanente* lorsqu'elle est définitive. Une faute permanente survient à un moment arbitraire dans le système, ensuite, elle continue tout au long de l'exécution de l'algorithme. Un exemple d'une faute permanente est une défaillance matérielle du nœud.

— Une faute est dite *intermittente* si elle survient périodiquement et de façon répétitive tout au long de l'exécution de l'algorithme. Un exemple d'un faute intermittente est une perte de connexion répétitive dans un réseau.

— Une faute est *transitoire* si elle a une durée finie. Elle survient à un moment arbitraire dans le système et disparait au bout d'un temps fini. Par exemple, un message qui n'arrive pas à destination dans un réseau est une faute transitoire.

**Nature de la faute**

— Une faute *franche* survient lorsque les nœuds cessent définitivement d'exécuter leur algorithme. Un exemple d'une faute franche est un appareil défectueux qui arrête de fonctionner définitivement.

— En revanche, lorsqu'une faute *byzantine* survient, les nœuds continuent de fonctionner mais exhibent un comportement arbitraire et malveillant. Un faute byzantine peut être par exemple un virus ou une attaque extérieure.

— On parle d'une faute *sur état* lorsque l'état d'un nœud a été modifié suite à une ou plusieurs fautes. Par exemple, la corruption ou le dysfonctionnement de la mémoire d'un nœud.

— Un faute *par omission* survient lorsqu'un nœud omet de communiquer avec les autres nœuds du système. Par exemple, on parle d'une faute par omission lorsqu'un message est perdu.

De nombreuse techniques ont été conçues pour gérer la présence de fautes dans les systèmes distribués, nous pouvons citer l'évitement de fautes, la suppression de faute et la *tolérance aux fautes*. Nous nous intéressons dans le contexte de cette thèse à la tolérance aux fautes. La tolérance aux fautes consiste à considérer que des fautes transitoires sont inévitables dans un réseau, en conséquence, il faut concevoir des systèmes qui continuent de fonctionner correctement en réparant ou en tolérant ces fautes. Autrement dit, la tolérance aux fautes est une méthode qui permet à un système de fonctionner correctement (éviter qu'il ne tombe définitivement en panne) malgré la survenue de fautes dans le système.

Il existe plusieurs approches tolérantes aux fautes dans la littérature. D'une part, l'approche *robuste* qui consiste à résister aux pannes survenues dans le système en utilisant par exemple des techniques de redondance et de réplication afin que le système conserve un comportement légitime. D'autre part, une approche tolérante aux fautes transitoires dite *auto-stabilisante* qui consiste à récupérer sur des fautes transitoires survenues dans un système en un temps fini et sans intervention extérieure. L'approche auto-stabilisante est au coeur de cette thèse.

## 2.3 Auto-stabilisation

De nombreux résultats théoriques auto-stabilisants ont été présentés dans la littérature ces dernières années. Dans les prochains chapitres, nous allons longuement décrire les réseaux de capteurs ainsi que leur contraintes. Les systèmes auto-stabilisants répondent en partie aux problèmes liés à de tels réseaux tels que : la défaillance des nœuds (batterie, panne) et le dynamisme du réseau (changement fréquent de topologie). Les capteurs qui fonctionnent de manière autonome doivent être capables de reconstituer la communication dans le réseau sans aucune intervention externe. En d'autres termes, le protocole de routage qui prend en charge un réseau de capteurs doit être auto-stabilisant. Dans la suite, nous allons d'abord définir un système auto-stabilisant puis nous définirons plus formellement l'auto-stabilisation.

### 2.3.1 Définitions

Le paradigme de l'auto-stabilisation a été introduit par Edgar Djikstra en 1974 [Dij74]. L'auto-stabilisation est la capacité d'un système à retrouver une configuration légitime à partir d'une configuration initiale quelconque et ceci sans intervention extérieure et en un temps fini. Un système auto-stabilisant est un système qui garantit deux propriétés :

1. **La convergence :** La convergence garantit que le système auto-stabilisant atteindra un état légitime quelque soit son état initial.
2. **La clôture :** La clôture garantit, qu'en l'absence de fautes transitoires, le système dans un état légitime conserve un état légitime.

**Définition 1 (Spécification)** *Considérons $\mathcal{P}$ un problème à résoudre. Une* spécification *de $\mathcal{P}$ est un prédicat qui est satisfait par chaque algorithme qui résout $\mathcal{P}$.*

Après avoir définit ce qu'est une spécification. Nous allons donner une définition formelle de l'auto-stabilisation

**Définition 2 (Auto-stabilisation [Dij74])** *Considérons $\mathcal{P}$ un problème, et $\mathcal{S}_\mathcal{P}$ une spécification de $\mathcal{P}$. Un algorithme $\mathcal{A}$ est auto-stabilisant pour $\mathcal{S}_\mathcal{P}$ si et seulement si pour chaque configuration $\gamma_0 \in \Gamma$ et pour chaque execution $\epsilon = \gamma_0\gamma_1\ldots$, il existe un préfixe fini $\gamma_0\gamma_1\ldots\gamma_l$ de $\epsilon$ tel que chaque exécution de $\mathcal{A}$ commençant de $\gamma_l$ satisfait $\mathcal{S}_\mathcal{P}$.*

Il y a eu plusieurs travaux dans la littérature qui proposent des solutions pour obtenir un algorithme auto-stabilisant à partir d'un algorithme distribué. Les auteurs dans [KP93] ont introduit une couche auto-stabilisante qui ré-initialise le système de façon auto-stabilisante si une incohérence est détectée afin de garantir au système de converger vers une configuration légitime. Cette solution est généralement nommée le stabilisateur car elle utilise un nœud particulier, par exemple une racine élue, pour examiner tout le système (ses variables et les variables des autres nœuds) et détecter une incohérence. Cependant, cette solution est coûteuse en termes d'espace mémoire, de coût en messages et de temps de stabilisation. Les auteurs dans [APSVD94] ont proposé une amélioration de [KP93] qui consiste à remplacer la détection centralisée (par l'élu) de l'incohérence par une détection locale au niveau de chaque nœud.

Une autre solution proposée par [DIM90] fait une composition modulaire de plusieurs algorithmes auto-stabilisants. Les auteurs ont prouvé qu'il est possible que plusieurs algorithmes auto-stabilisants s'exécutent en même temps sur un système et composent ainsi

un seul algorithme auto-stabilisant. Cette technique est souvent utilisée de nos jours dans la littérature. Par ailleurs, un travail dans cette thèse utilise cette technique et propose un algorithme auto-stabilisant composé de plusieurs algorithmes pour construire un arbre couvrants de diamètre minimum (voir chapitre 3).

### 2.3.2 Modèle

Tout système distribué possède des caractéristiques qui définissent son fonctionnement. Pour modéliser un système distribué, il faut définir l'ensemble des caractéristiques de ce système, par exemple : les nœuds sont-ils anonymes ou non, un nœud à t-il un rôle particulier (réseau non uniforme), quel est le modèle de communication entre les nœuds et enfin le système est-il synchrone ou asynchrone.

#### Modèle de communication

Nous modélisons notre système comme un graphe non-orienté $G = (V, E)$ où $V$ est un ensemble de nœuds et $E$ la relation binaire qui désigne la capacité de deux nœuds à communiquer. Autrement dit, $(u, v) \in E$ si et seulement si $u$ et $v$ sont capables de communiquer entre eux. L'ensemble des nœuds capables de communiquer avec le nœud $v$ sont appelés les *voisins* de $v$, l'ensemble des voisins de $v$ est noté par $N_v$. Afin d'effectuer des calculs distribués et de réaliser une tâche donnée, les nœuds ont besoins d'échanger des informations. Deux modèles de communications majeurs existent dans la littérature : le modèle de communication par passage de messages et le modèle de communication par lecture de registre. L'utilisation de l'un ou l'autre des modèles de communication n'est pas restrictive dans la mesure ou des transformateurs existent dans la littérature [Dol00] pour passer d'un modèle de communication à un autre dans le cas de graphes non orientés.

Dans le modèle *par passage de messages*, les nœuds communiquent directement entre eux en échangeant des messages. Ces messages sont envoyés par les nœuds dans le médium de communication. Il existe plusieurs variantes de ce modèle en fonction des hypothèses sur le médium de communication. Ces hypothèses peuvent être émises sur la bi-directionalité du médium, sa fiabilité et l'ordre de délivrance de messages.

Dans *le modèle à états*, appelé aussi modèle à partage de registre [Dol00], les communications entre voisins sont modélisées par une lecture directe des variables au lieu d'échange de messages. Chaque nœud a un ensemble de variables partagées. Un nœud peut lire et écrire dans son propre registre mais il peut uniquement lire les registres de ses voisins. Nous considérons ici q'un nœud effectue ses trois tâches, à savoir (i) lecture des registres de ses voisins, (ii) execution de son algorithme, (iii) mise à jour de son registre, en une seule étape atomique. Dans cette thèse, nous utilisons le modèle à états.

L'état $\gamma(v)$ d'un nœud $v$ est le vecteur des valeurs de toutes les variables de $v$ à un moment donné. Une configuration du réseau est l'ensemble des états de tous les nœuds du réseau à un moment donné. $\Gamma$ désigne l'ensemble de toutes les configurations du système.

Pour décrire clairement le fonctionnement de notre algorithme, nous avons utilisé la description de [Dij74] pour présenter ces règles. En effet, l'algorithme de chaque nœud est un ensemble fini de *règles*. Chaque règle consiste en une étiquette, une garde et une action présentée comme suit :

$$<\textit{étiquette}> :<\textit{garde}> \longrightarrow <\textit{action}>.$$

*L'étiquette* d'une règle est simplement un nom pour se référer à l'action dans le texte.

*La garde* d'une règle dans l'algorithme de $v$ est un prédicat booléen impliquant les variables de $v$ et ses voisins. Si ce prédicat est vrai alors le nœud est dit activable dans la configuration courante. Une règle peut être exécutée uniquement si le nœud est activable.

*L'action* d'une règle de $v$ met à jour une ou plusieurs variables de $v$ et par conséquent modifie le registre de ce nœud. L'action d'une règle décrit les changements qui devront s'effectuer sur le registre d'un nœud si ce dernier est activé par le démon. Les actions d'une règle mettent à jour une ou plusieurs variables de $v$.

### Asynchronisme

Un système distribué est dit *synchrone* si le temps de transfert de l'information (lecture d'un registre, échange d'un message) d'un nœud à un autre est le même pour tout le système. Un système distribué est dit *Semi-synchrone* si le temps de transfert d'une information d'un nœud à un autre est différent mais une borne supérieur sur ce temps est connue. Dans le cas d'un système distribué *Asynchrone*, le temps de transfert d'information d'un nœud à l'autre est fini mais non borné.

L'asynchronisme du système est modélisé par un *démon*, appelé aussi *adversaire* car il met en difficulté l'execution de l'algorithme. Un nœud est activable lorsqu'il peut effectuer une action dans son algorithme. L'adversaire est un dispositif indépendant des nœuds, il possède une vision globale et choisit, à chaque étape, le sous-ensemble de nœuds parmi les nœuds activables qu'il autorise à executer leur commande. Dans ce cas là un nœud est dit activé. Suivant son équité, un démon choisit parmi l'ensemble des nœuds activables (ceux qui peuvent exécuter une commande) ceux qu'il active. Selon cette taxonomie [DT11], il est important de distinguer 2 classifications majeurs des démons en fonction de leurs caractéristiques : l'équité et la distribution.

### Distribution

Un démon est dit *central* s'il choisit un seul nœud activable pour exécuter une règle à chaque étape de l'algorithme. Par ailleurs, un démon est dit *distribué* lorsqu'il choisit un ou plusieurs nœuds activables pour exécuter une règle à chaque étape de l'algorithme.

### Equitée

Un démon qui choisit tous les nœuds activables à chaque exécution d'une étape de l'algorithme est dit *équitable (système synchrone)*. Un démon est dit *faiblement équitable* si un nœud est activable infiniment souvent, il sera éventuellement activé à une étape de l'algorithme. Dans le cas d'un système totalement *asynchrone*, le démon ne fait aucune hypothèse sur le choix des nœuds activables pour exécuter une règle de l'algorithme. On parle alors de démon *non équitable* ou *inéquitable*.

### Anonymat

Dans le cas d'un système distribué *Non anonyme*, les nœuds possèdent tous des identifiants deux à deux distincts. Dans cette thèse, nous considérons uniquement des systèmes

non anonymes (identifiés). Plus précisément, nous considérons qu'il existe un unique identifiant $\text{ID}_v$ pour chaque nœud $v$ pris dans l'ensemble $[0, n^c]$ pour une certaine constante $c$. Dans la suite, nous désignons $n$ comme le nombre de nœuds du réseau (taille du réseau). Dans un système distribué *Anonyme*, les nœuds ne possèdent aucun identifiant.

### Uniformité

Un système distribué est dit *uniforme* lorsque tous les nœuds exécutent le même algorithme distribué.

Un système distribué est *semi-uniforme* si tous les nœuds excepté un (ou plusieurs) exécutent le même algorithme distribué. C'est le cas par exemple des systèmes avec un nœud qui représente la racine (potentiellement un élu) qui se comporte différemment des autres nœuds pour la construction d'un arbre couvrant.

Dans un système distribué *non uniforme*, les nœuds n'exécutent pas tous le même algorithme. Pour un tel système, il existe au moins un nœud qui exécute un algorithme différent.

### Complexité algorithmique

La complexité algorithmique est une méthode utilisée pour mesurer les performances d'un algorithme. Il existe essentiellement deux complexités dans le cadre du modèle à états, la complexité temporelle et la complexité spatiale :

Il existe deux manières de calculer la complexité temporelle d'un algorithme distribué, la plus utilisée est la notion de *ronde*. Certains résultats proposent un calcul en *pas de calcul* où un pas de calcul est une activation d'un nœud.

On considère l'ensemble des nœuds activables $A(\gamma)$ pour une configuration $\gamma$ dans un système asynchrone. Une ronde est terminée si quelque soit $v \in A(\gamma)$, $v$ est activé par le démon ou est désactivé par l'activation de l'un de ses voisins. Aucune restriction n'est faite, sur le nombre de fois qu'un nœud peut être activé avant la fin de la ronde.

La complexité en espace mémoire est le nombre de bits nécessaires pour stocker les variables du registre d'un nœud durant l'execution de l'algorithme.

**Algorithme auto-stabilisant silencieux** Dans le modèle à états ou celui à registres partagés, un algorithme auto-stabilisant est silencieux [DGS96] si, après la stabilisation de l'algorithme, les variables des registres restent inchangées. Dans un modèle à passage de messages, un algorithme est dit silencieux s'il n'y a plus de circulation de messages, ou si le contenu des messages échangés ne change pas après convergence.

Dolev, Gouda et Shneider [DGS96] ont prouvé que, dans un modèle à registres, la mémoire minimum requise par un algorithme auto-stabilisant silencieux de construction d'arbre couvrant où d'élection est $\Omega(\log n)$ bits sur chaque nœud. Dans le cadre non-silencieux, Beauquier, Gradinariu et Johnen [BGJ99] ont prouvé qu'il n'était pas possible de faire de telles tâches en utilisant un nombre constant de bits. Blin et Tixeuil [BT13] ont prouvé qu'il était possible de faire une élection et une construction d'un arbre couvrant dans un anneau en utilisant $O(\log \log n)$ bits par nœud. La Figure 2.1 représente la courbe d'un algorithme silencieux comparée à la courbe d'un algorithme bavard.

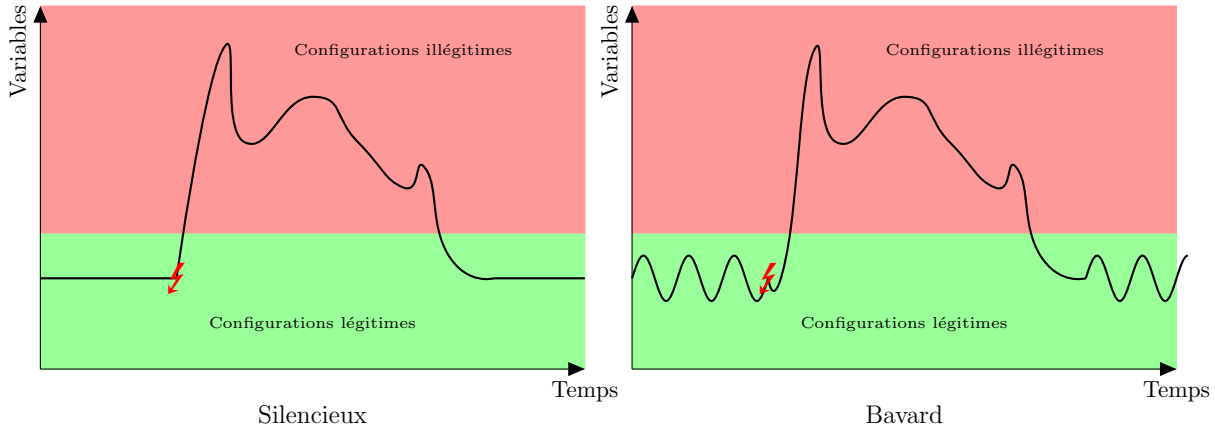



FIGURE 2.1 – Algorithme silencieux et algorithme bavard

## 2.4 Arbres couvrants sous contraintes

Dans un réseau informatique, les échanges d'informations doivent être optimisés afin d'éviter la surcharge des liens de communication. Une structure de communication adéquate pour répondre à ce problème est l'arbre couvrant. Un arbre couvrant est une structure qui permet de minimiser le nombre de liens de communication utilisés en maintenant un unique chemin entre toutes paires de nœuds. Il n'existe pas un arbre couvrant mais une multitude d'arbre couvrants pour un réseau donné. Le type d'arbre couvrant construit à partir d'un réseau dépend de la métrique à favoriser. Il peut s'agir par exemple de minimiser le délai de communication entre les nœuds, de minimiser le degré des nœuds ou de minimiser le coût de construction dans le cas d'un arbre pondéré. Dans cette section, nous donnerons d'abord quelques notions sur les graphes et les arbres couvrants, ces définissions seront utiles pour les chapitres suivants de la thèse. Ensuite, nous décrirons plus en détails les types d'arbres couvrants. Puis, nous dresserons un état de l'art non exhaustif sur les solutions auto-stabilisantes pour la construction d'arbres couvrants.

### 2.4.1 Notions sur les arbres couvrants

Dans cette sous-section, nous donnons quelques notions sur les arbres couvrants.

**Définition 3 (Graphe)** *Un graphe $G = (V, E)$ non orienté est constitué de deux entités : $V$ un ensemble fini non vide d'éléments, les éléments de $V$ sont appelés sommets ou nœuds, et $E$ un ensemble tel que $E \subset V \times V$, les éléments de $E$ sont appelés arêtes ou liens.*

Pour un graphe G, on note $n = |V|$ le nombre de sommets du graphe et $m = |E|$ le nombre d'arêtes du graphe.

**Définition 4 (Sous-graphe)** *Le graphe $G' = (V', E')$ est un sous-graphe de $G$ lorsque $V' \subseteq V$ et $E' \subseteq E$.*

**Définition 5 (Graphe partiel)** *Le sous-graphe $G' = (V', E')$ est dit graphe partiel de $G$ si et seulement si $V' = V$ , $E' \subseteq E$ et $G'$ est connexe.*

**Définition 6 (Chemin)** *Soit une paire de sommets $u, v \in V$. Un chemin de $u$ à $v$, noté $P(u, v)$, est une séquence de sommets $< u, v_0, \ldots, v_k, v >$ tel que $\forall i, 0 \geq i < k$ et il existe une arête entre les sommets $v_i$ et $v_{i+1}$ dans $G$. De plus, le chemin $P(u, v) =< u, v_0, \ldots, v_k, v >$ est dit de longueur $k + 2$.*

**Définition 7 (Graphe connexe)** *Le graphe $G = (V, E)$ est dit connexe si et seulement si il existe au moins un chemin entre toutes paires de sommets $u, v \in V$ .*

**Définition 8 (Cycle)** *Un cycle $C$ est un chemin $P(u, v)$ tel que $v = u$.*

**Définition 9 (Plus court chemin)** *Soit un graphe connexe $G = (V, E)$ et une paire de sommets $u, v \in V$. Le plus court chemin entre $u$ et $v$ est le chemin de longueur minimal parmi l'ensemble de tous les chemins entre $u$ à $v$.*

**Définition 10 (Distance)** *Soit un graphe connexe $G = (V, E)$ et $u, v \in V$ deux sommets de $G$. La distance $d(u, v)$ entre les sommets $u$ et $v$ dans $G$ est la longueur du plus court chemin entre les deux sommets.*

**Définition 11 (Excentricité)** *Soit un graphe connexe $G = (V, E)$ et $ecc(v)$ l'excentricité d'un nœud $v \in V$. L'excentricité de $v$ est égale à la distance maximale entre $v$ et tout nœud $u \in V$. En d'autres termes : $ecc(v) = \max\{d(u, v) : u \neq v\}$.*

**Définition 12 (Centre(s) du graphe)** *Le ou les centre(s) d'un graphe est l'ensemble de ses sommets d'excentricité minimale.*

**Définition 13 (Diamètre)** *Soit un graphe connexe $G = (V, E)$ et $D$ le diamètre de $G$. Le diamètre du graphe $G$ est égale à la plus grande distance entre toutes paires de sommets $u, v \in V$. Autrement dit le diamètre est l'excentricité maximale de tout nœud $v \in V$, $D = \max\{ecc(v) : v \in V\}$.*

**Définition 14 (Voisins d'un nœud)** *Soit un nœud $v \in V$. L'ensemble des nœuds $u$ tel que $\{u, v\} \in E$ sont appelés voisins de $v$.*

**Définition 15 (Degré d'un nœud)** *Soit un nœud $v \in V$. Le degré de $v$, noté $d(v)$ est le nombre de ses voisins $u$.*

**Définition 16 (Degré du graphe)** *Le degré de $G$, noté $\Delta$ est égal au plus fort degré de l'ensemble des sommets de $G$, autrement dit $\Delta = \max\{d(v) : v \in V\}$.*

**Définition 17 (Arbre)** *Soit $T = (V, E)$ un graphe. Les définitions suivantes sont équivalentes pour la définition d'un arbre :*

- *$T$ est connexe et sans cycle ;*
- *Il existe un seul et unique chemin entre toutes paires de sommets $u, v \in V$ ;*

— *$T$ est connexe, et $T$ n'est plus connexe si une arête de $T$ est supprimée ;*
— *$T$ est sans cycle, et on crée un cycle et un seul cycle en ajoutant une arête entre deux sommets non- adjacents dans $T$ ;*
— *$T$ est connexe et contient $n-1$ arêtes ;*
— *$T$ est un sous-graphe acyclique et contient $n-1$ arêtes.*

**Définition 18 (Sous-arbre)** *Soit $T = (V_T, E_T)$ et $T' = (V_{T'}, E_{T'})$ deux arbres. $T'$ est sous-arbre de $T$ si et seulement si $V_{T'} \subseteq V_T$ et $E_{T'} \subseteq E_T$.*

**Définition 19 (Arbre couvrant)** *Un arbre $T = (V_T, E_T)$ est un arbre couvrant le graphe $G$ si et seulement si $T$ est un graphe partiel de $G$.*

**Définition 20 (Arborescence)** *Une arborescence (ou arbre enraciné) $T = (V_T, E_T)$ est un graphe connexe orienté sans cycle admettant un sommet distingué $r \in V_T$ appelé racine, et tel que pour tout autre sommet $v \in V_T$, il existe un chemin unique allant de $v$ vers $r$ (ou le contraire).*

**Définition 21 (Feuille)** *Soit un arbre $T = (V_T, E_T)$, un nœud $v \in V_T$ est une feuille si $v$ a un seul voisin ($d(v) = 1$).*

**Définition 22 (Sommet interne)** *Soit un arbre $T = (V_T, E_T)$, un nœud $v \in V_T$ est un sommet interne si $v$ n'est pas une feuille.*

**Définition 23 (Parent)** *Soit $T = (V_T, E_T)$ un arbre enraciné au sommet $r \in V_T$, le sommet $u \in V_T$ est le parent du sommet $v \in V_T$ si $u$ est l'unique voisin de $v$ sur le chemin entre $v$ et la racine $r$ dans l'arbre $T$.*

**Définition 24 (Enfants)** *Soit $T = (V_T, E_T)$ un arbre enraciné au sommet $r \in V_T$, tous les sommets voisins de $v \in V_T$ qui ne sont pas parent de $v$ sont ses enfants dans $T$.*

**Définition 25 (Descendants)** *Les descendants de $v$ dans $T$ sont tous les sommets présents dans le sous-arbre enraciné en $v$.*

**Définition 26 (Rang et Hauteur)** *Soit un arbre $T = (V_T, E_T)$, le rang d'un sommet $v$ est sa distance à la une racine $r$ tel que $r$ est de rang 0. Le sommet $v$ et un sommet $u$ à la même distance de la racine sont dits de même rang. La hauteur de $T$ est le rang maximum.*

### 2.4.2 Algorithmes auto-stabilisants de construction d'arbres couvrants sous-contraintes

La construction d'arbres couvrants sous-contraintes a été largement étudiée dans le cadre des systèmes distribués ainsi que dans les systèmes auto-stabilisants. Il existe de nombreuses métriques d'arbres couvrants sous-contraintes, en voici une liste non exhaustive de constructions auto-stabilisantes d'arbres couvrants, pour une description plus détaillée voir [Rov09, Bli11], le tableau 2.2 représente un résumé des travaux sur ces arbres couvrants.

| Problème | Articles | Semi-uniforme | Anonyme | *Connaissance* | *Communications* | *Adversaire* | *Equité* | *Atomicité* | Silencieux | *Espace mémoire* | *Temps de convergence (rondes)* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| BFS | [DIM90] | ✓ | ✓ | | R | C | $f$ | $\oplus$ | | $O(\Delta \log n)$ | $O(D)$ |
| | [AG90] | | | $n$ | R | C | | | | $O(\log n)$ | $O(n^2)$ |
| | [AKY91] | | | | R | D | $f$ | $\oplus$ | | $O(\log n)$ | $O(n^2)$ |
| | [HC92] | ✓ | ✓ | $n$ | R | D | $I$ | | | | |
| | [Dol93] | | | | R | C | $f$ | $\oplus$ | | $O(\Delta n \log n)$ | $\Theta(D)$ |
| | [ABB98] | | | B [1] | M | D | | | | $O(\log n)$ | $O(n)$ |
| | [DLV08] | | | | R | D | $f$ | $\oplus$ | | $O(\log n)$ | $O(n)$ |
| | [CRV11] | ✓ | | | R | D | $I$ | $\oplus$ | | $O(\log n)$ | $O(D^2)$ |
| DFS | [HC93] | ✓ | ✓ | $n$ | R | D | | | | $O(\log \Delta n)$ | |
| | [CD94] | ✓ | ✓ | | R | C | $f$ | $\oplus$ | | $O(n \log \Delta)$ | $O(Dn\Delta)$ |
| | [HW97] | | ✓ | $n$ | R | C | $f$ | | X | $O(\log n)$ | |
| | [DJPV00] | ✓ | ✓ | | R | D | $f$ | | | $O(\log \Delta)$ | $O(Dn\Delta)$ |
| SPT | [HL02] | ✓ | ✓ | | R | C | $I$ | | | $O(\log n)$ | |
| | [JT03] | ✓ | ✓ | | R | D | $f$ | | | $O(\log n)$ | |
| | [GS03] | ✓ | ✓ | | M/R | D | | | | $O(\log n)$ | $O(D)$ |
| | [BK07] | | | $D$ | M | D | | $\oplus$ | | $O(\log^2 n)$ | $O(D)$ |
| MST | [GS99, GS03] | | | $n$ | M | D | | | ✓ | $\Theta(n \log n)$ | $\Omega(n^2)$ |
| | [HL01] | | | $O(D)$ | M | D | | | $X$ | $O(\log n)$ | $O(n^3)$ |
| | [BPRT09, BPRT16] | | | | R | D | $I$ | | X | $O(\log n)$ | $O(n^3)$ |
| | [BDPBR10] | | | | R | D | $I$ | | X | $\Omega(\log^2 n)$ | $O(n^2)$ |
| | [KKM11] | | | | R | D | $I$ | | X | $O(\log n)$ | $O(n)$ |
| | [BF15] | | | | R | D | $I$ | | ✓ | $O(\log^2 n)$ | $Poly$ |
| MDegST | [BPR09a, BPBR11] | | | | M | D | | $\oplus$ | ✓ | $O(\Delta \log n)$ | $O(mn^2 \log n)$ |
| | [BF15] | | | | R | D | $I$ | | ✓ | $O(\log n)$ | $Poly$ |
| MDiamST | [BLB95] | | ✓ | | M | D | $I$ | | ✓ | $O(n^2 \log n + n \log W)$ | $O(n\Delta + d^2 + n \log^2 n)$ |
| Steiner | [BPR09b, BPR13] | | | | M | D | $I$ | $\oplus$ | ✓ | $O(\Delta \log n)$ | $O(zD)$ |

FIGURE 2.2 – Algorithmes auto-stabilisants asynchrones pour la construction d'arbres couvrants.
$D$ : diamètre du réseau ; $\Delta$ : degré maximum du réseau ; $z$ : nombre de nœuds dans l'arbre de Steiner ; m1 : poids minimum dans le réseau ; m2 : poids maximum dans le réseau ; R : registres partagés ; M : passage de messages ; Adversaire : Distribué (D), central (C), inéquitable ($I$), faiblement équitable ($f$), fortement équitable ($F$). Atomicité : $\oplus$ lecture ou écriture. ST : arbre couvrant ; DFS : arbre en profondeur d'abord ; BFS : arbre en largeur d'abord ; MDegST : arbre couvrant de degré minimum ; MST : arbre de poids minimum ; SPT : arbre de plus court chemin. MDiamST : arbre de diamètre minimum.

### Arbre couvrant en largeur d'abord

Dolev, Israeli et Moran [DIM90, DIM93] sont parmi les premiers à avoir proposé un algorithme auto-stabilisant de construction d'*arbre couvrant en largeur d'abord* (BFS pour « Breadth First Search » en anglais). Il considère le modèle de communication par registres, avec un démon centralisé, et propose un algorithme semi-uniforme. Cet algorithme, qui fonctionne par propagation de distance, est une brique de base pour la conception d'un algorithme auto-stabilisant pour le problème de l'exclusion mutuelle dans un réseau asynchrone, anonyme, et dynamique.

Afek, Kutten et Yung [AKY91] ont proposé quand à eux un algorithme uniforme construisant un BFS dans un réseau non-anonyme. La racine de l'arbre couvrant est le

nœud d'identifiant maximum. Chaque nœud met à jour sa variable racine. Les configurations erronées vont être éliminées grâce à cette variable. Dès qu'un nœud s'aperçoit qu'il n'a pas la bonne racine, il commence par se déclarer racine lui même, et effectue ensuite une demande de connexion en inondant le réseau. Cette connexion sera effective uniquement après accusé de réception par la racine (ou par un nœud qui se considère de façon erronée comme une racine). Afek et Bremler-Barr [ABB98] ont amélioré l'approche proposée dans [AKY91]. Dans [AKY91], la racine élue pouvait ne pas se trouver dans le réseau car la variable racine peut contenir un identifiant maximum erroné après une faute. Dans [ABB98], la racine est nécessairement présente dans le réseau.

Datta, Larmore et Vemula [DLV08] ont proposé un algorithme auto-stabilisant reprenant l'approche de Afek et Bremler-Barr [ABB98]. Ils construisent de manière auto-stabilisante un BFS afin d'effectuer une élection. Pour ce faire, ils utilisent des vagues de couleurs différentes afin de contrôler la distance à la racine, ainsi qu'un mécanisme d'accusé de réception afin d'arrêter les modifications de l'arbre. C'est donc en particulier un algorithme silencieux [DGS96].

Arora et Gouda [AG90, AG94] ont présenté un système de « réinitialisation » après faute, dans un réseau non anonyme. Ce système en couches utilise trois algorithmes : un algorithme d'élection, un algorithme de construction d'arbre couvrant, et un algorithme de diffusion. Les auteurs présentent une solution silencieuse et auto-stabilisante pour chacun des trois problèmes. Comme un certain nombre d'autres auteurs par la suite ([HL01, BK07]), ils utilisent la connaissance à priori d'une borne supérieure sur le temps de communications entre deux nœuds quelconques dans le réseau afin de pouvoir éliminer les cycles résultants d'une configuration erronée après une faute.

Huang et Chen [HC92] ont proposé un algorithme semi-uniforme de construction auto-stabilisante de BFS. Leur contribution la plus importante reste toutefois les nouvelles techniques de preuves d'algorithmes auto-stabilisants qu'ils proposent dans leur article.

Dans un cadre dynamique, Dolev [Dol93] a proposé un algorithme auto-stabilisant de routage, et un algorithme auto-stabilisant d'élection. Pour l'élection, chaque nœud devient racine d'un BFS. La contribution principale est le temps de convergence de chaque construction de BFS, qui est optimal en $O(D)$ rondes où $D$ est le diamètre du graphe. De manière indépendante, Aggarwal et Kutten [AK93] ont proposé un algorithme de construction d'un arbre couvrant enraciné au nœud de plus grand identifiant, optimal en temps de convergence, $O(D)$ rondes.

Enfin, Cournier, Rovedakis et Villain [CRV11] ont proposé un algorithme auto-stabilisant convergent en $O(D^2)$ rondes, et plus précisément $O(n^6)$ pas de calcul, où une étape et le nombre de fois que s'exécute un algorithme.

### Arbre couvrant en profondeur d'abord

Collin et Dolev [CD94] reprennent la même approche que Dolev, Israeli et Moran [DIM90, DIM93] afin de concevoir un algorithme de construction d'arbres couvrants en profondeur d'abord (DFS, pour « Depth Fisrt Search » en anglais). Pour cela, chaque nœud $u$ connaît le numéro de port de chaque arête $e = \{u, v\}$ incidente à $u$, ainsi que le numéro de port de $e$ en son autre extrémité $v$. Avec cette connaissance, un ordre lexicographique est créé pour construire un parcours DFS.

La construction auto-stabilisante d'arbres couvrants en profondeur d'abord va souvent de paire dans la littérature avec le parcours de jeton. Il est important de noter que, qui

dit circulation de jeton dit algorithme non-silencieux car le jeton transporte une information qui évolue le long du parcours. Huang et Chen [HC93] ont proposé un algorithme auto-stabilisant pour la circulation d'un jeton dans un réseau anonyme semi-uniforme. Le jeton suit un parcours en profondeur aléatoire. Huang et Wuu [HW97] ont proposé un autre algorithme auto-stabilisant pour la circulation d'un jeton, cette fois dans un réseau anonyme uniforme. Ces deux algorithmes nécessitent une connaissance à priori de la taille du réseau. Contrairement à ces deux algorithmes, l'algorithme de Datta, Johnen, Petit et Villain [DJPV00] ne fait aucune supposition à priori sur le réseau. De plus, cet algorithme améliore la taille mémoire de chaque nœud en passant de $O(\log n)$ bits à $O(\log \Delta)$ bits, où $\Delta$ est le degré maximum du réseau.

### Arbre couvrant de plus court chemin

Le problème de l'arbre couvrant de plus court chemin (SPT pour « Shortest Path Tree » en anglais) est la version pondérée du BFS. Dans ce cadre, Huang et Lin [HL02] ont proposé un algorithme semi-uniforme auto-stabilisant pour ce problème. Chaque nœud calcule sa distance par rapport à tous ses voisins comme le fait l'algorithme de Dijkstra. Johnen et Tixeuil [JT03] ont proposé deux algorithmes auto-stabilisants de construction d'arbres couvrants, dans un cadre dynamique, où le poids des arêtes peut changer au cours du temps. Le principal apport de leur approche est de s'intéresser à la propriété sans-cycle introduite par [GB81]. Cette propriété stipule que l'arbre couvrant doit s'adapter aux changements de poids des arêtes sans se déconnecter ni créer de cycle. Gupta et Srimani [GS03] supposent le même dynamisme que Johnen et Tixeuil [JT03]. Ils ont proposé plusieurs algorithmes auto-stabilisants, dont un algorithme semi-uniforme construisant un arbre SPT. Le principal apport de cette dernière contribution est de fournir un algorithme auto-stabilisant silencieux, optimal en espace et en temps de convergence.

Burman et Kutten [BK07] se sont intéressés à un autre type de dynamisme : l'arrivée et/ou le départ des nœuds, et/ou du arêtes du réseau. De plus, ces auteurs ont proposé d'adapter l'atomicité lecture/écriture du modèle par registre au modèle par passage de message. Ce nouveau concept est appelé *atomicité envoi/réception* (« send/receive atomicity »).

### Arbre couvrant de poids minimum

La construction d'un arbre couvrant de poids minimum (appelé *MST* pour Minimum Spanning Tree) considère un graphe pondéré. Le but d'un MST est de minimiser le coût de construction de la structure. Autrement dit, un arbre couvrant de poids minimum est un arbre couvrant dont la somme des poids et des arêtes est minimum. Gupta et Srimani [GS99, GS03] ont proposé le premier algorithme auto-stabilisant pour la construction d'un MST, leur algorithme se base sur l'algorithme de Dijkstra de calcul de plus court chemin entre toutes paires de nœuds. Cet algorithme est proposé dans un modèle de communication par passage de messages dans un réseaux semi-synchrone. Dans le même cadre, Higham et Lyan proposent un algorithme qui utilise la propriété qui dit que l'arête de poids maximum d'un cycle ne fait pas partie de l'arbre couvrant de poids minimum. Leur algorithme utilise une exploration perpétuelle, il est donc non silencieux.

Blin, Potop-Butucaru, Rovedakis et Tixeuil [BDPBR10] ont proposé le premier algorithme auto-stabilisant pour la construction d'un MST qui ne fait aucune hypothèse sur une connaissance de la taille du réseau, il fonctionne dans un réseau asynchrone. De

plus, il garantit la propriété sans-cycle et il est le premier à atteindre une taille mémoire de $O(\log n)$ bits par nœud. Cet algorithme utilise une circulation de jeton, il est donc non-silencieux.

Blin, Dolev, Potop-Butucaru et Rovedakis proposent [BDPBR10] le premier algorithme auto-stabilisant utilisant un schéma d'étiquetage, à savoir des étiquettes pour trouver l'ancêtre commun dans l'arbre de deux nœuds. Grâce à leur méthode, ils proposent un algorithme auto-stabilisant non-silencieux améliorant le ratio temps de convergence espace mémoire par nœud.

Korman, Kutten et Masuzawa [KKM11] améliorent ce ratio en proposant un algorithme auto-stabilisant non-silencieux atteignant $O(n)$ rounds. A cet effet, ils utilisent des trains d'informations.

Il est important de noter qu'à part le premier algorithme cité, les trois autre algorithmes sont non-silencieux. Blin et Fraigniaud [BF15] propose un algorithme auto-stabilisant silencieux optimal en espace mémoire, à savoir $O(\log^2 n)$ bits pas nœud, tout en restant polynomial en nombre de rondes. Cet algorithme, converge en premier lieu vers un arbre couvrant, puis tout en maintenant la propriété sans-cycle vers un MST. Une des principales techniques utilisées est la preuve par étiquetage (*Proof labeling scheme*).

#### Arbre couvrant du plus court chemin entre toutes paires de nœuds

Appelé *APSP* pour All Pairs Shortest Path Spanning Tree, l'arbre couvrant de plus court chemin entre toutes paires de nœuds est une variante du SPT puisque il est recherché le plus court chemin entre toutes paires de nœuds et non entre un nœud distingué (la racine) et les autres nœuds du graphe. Les travaux [GBS00, Hua05] ont proposé des algorithmes auto-stabilisants pour la construction d'arbres couvrants de plus court chemin entre toutes paires de nœuds.

#### Arbre de Steiner

Le problème de la construction d'un *Arbre de Steiner* est une extension du problème de la construction d'un arbre couvrant de poids minimum (MST). Construire un arbre Steiner revient à construire un arbre couvrant MST uniquement pour un sous ensemble donné de nœuds du graphe. Le problème de la construction d'un arbre Steiner est un problème NP-complet. Blin, Potop-Buturacu et Rovedakis [BPR09b, BPR13] sont les seuls à notre connaissance à proposer un algorithme auto-stabilisant pour la construction d'un arbre de Steiner. Le poids de l'arbre construit est une $\log(z)-$approximation par rapport à la solution optimale (avec $z$ le nombre de nœuds à interconnecter dans l'arbre de Steiner). Leur algorithme est super-stabilisant car il garantit que le système est auto-stabilisant et supporte le départ de liens ou de nœuds du réseau.

#### Arbre couvrant de degré minimum

Appelé *MDST* pour Minimum Degree Spanning Tree pour un arbre couvrant de degré minimum est un arbre couvrant dont le degré est minimum. Le premier algorithme auto-stabilisant pour la construction d'un $M$DST a été proposé par Blin, Gradinariu et Rovedakis [BPBR11]. Etant donné que le calcul d'un MDST est un problème NP-complet, une solution optimale n'existe pas, ils utilise les résultats de Fürer et Ravaghachari [FR94].

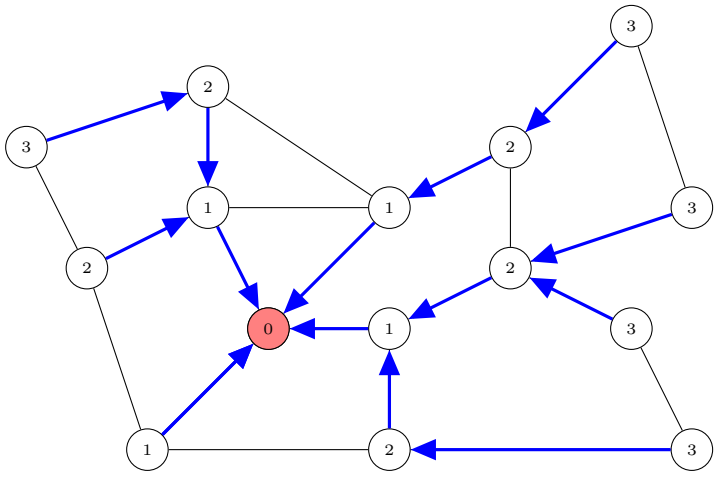



FIGURE 2.3 – Exemple d'arbre couvrant de diamètre minimum.

C'est pour cela que les auteurs ont proposé un algorithme d'approximation auto-stabilisant qui construit un arbre couvrant dont le degré maximum au plus $\delta + 1$ (avec $\delta$ : degré minimum du graphe). Cet algorithme converge en $O(\Delta \log n)$ rondes et utilise $O(mn^2 \log n)$ bits par nœud en mémoire.

Blin et Fraignaud ont proposé un algorithme auto-stabilisant silencieux optimal en espace mémoire pour le calcul d'un arbre couvrant de degré minimum, à savoir $O(\log^2 n)$ bits pas nœud, tout en restant polynomial en nombre de rondes. Cet algorithme propose un schéma d'étiquetages basé sur l'algorithme de Fürer et Ravaghachari [FR94].

### Arbre couvrant de diamètre minimum

La construction d'un arbre couvrant de diamètre minimum, appelé *MDiamST* pour Minimum Diameter Spanning Tree permet de minimiser le délai de communication entre toutes paires de nœuds dans le réseau. Autrement dit, un arbre couvrant de diamètre minimum minimise le diamètre du graphe (voir la sous-section 2.4.1 pour la définition d'un diamètre). Le problème du MDiamST est étroitement lié à la détermination des centres du système [HT95]. En effet, un centre est un nœud qui minimise son excentricité, autrement dit, sa distance à tout autre nœud du système. Ensuite, il est bien connu qu'un arbre couvrant BFS enraciné dans un des centres est un MDiamST.

Butelle, Lavault et Bui [BLB95] sont les seuls à avoir proposé un algorithme auto-stabilisant pour construire un arbre couvrant de diamètre minimum. Les auteurs partent du principe que calculer un arbre couvrant de diamètre minimum revient au calcul du centre d'un graphe. Une fois le centre du graphe trouvé, un arbre couvrant de plus court chemin enraciné au centre est construit.

La solution des auteurs pour construire un arbre couvrant de diamètre minimum est la suivante. Les auteurs considèrent un graphe pondéré où les poids des arêtes sont des nombres réels. Ils considèrent aussi un système anonyme, uniforme, asynchrone et un ordonnancer distribué. De plus, leur solution ne suppose aucune connaissance du réseau.

L'algorithme proposé par [BLB95] est composé de trois algorithmes eux-mêmes auto-stabilisants. Le premier algorithme attribue un identifiant unique et différent pour chaque nœud en utilisant un mécanisme de diffusion de labels dans le graphe. A la fin de l'execu-

tion, chaque nœud connait son identifiant et l'identifiant de chaque nœud dans le graphe. Le deuxième algorithme calcule le centre du graphe en utilisant un algorithme qui calcule le plus court chemin entre toutes paires de nœuds. Cela revient à utiliser l'algorithme classique de Dijkstra. Dans la dernière couche, un arbre couvrant du plus court chemin enraciné au centre calculé par la couche précédente est construit.

L'algorithme proposé par les auteurs calcule un arbre couvrant de diamètre minimum en $O(n\Delta + D^2 + n \log\log n)$ rondes. Il utilise une mémoire de $O(n^2 \log n + n \log W)$ bits par nœud, avec $\Delta$ le degré maximum d'un nœud, $W$ le poids maximum d'une arête et $D$ le diamètre du réseau.

Les auteurs proposent en conclusion de leur article une optimisation de leur algorithme en introduisant une borne $B$ sur le diamètre du réseau. Leur nouvel algorithme construit un MDiamST en $O(n)$ rondes et utilise $O(\log B)$ bits de mémoire par nœud.

## 2.5 Conclusion

Dans ce chapitre, nous avons présenté plusieurs concepts sur les systèmes distribués et sur l'auto-stabilisation. Nous avons défini les caractéristiques des systèmes distribués et celles des systèmes auto-stabilisants. Nous avons aussi décrit les différents paramètres qui définissent le modèle d'un système distribué auto-stabilisant. Nous avons introduit dans la section 2.4 des notions sur les arbres couvrants. Nous avons décrit les différents types d'arbres couvrants et citer quelques travaux majeurs sur les algorithmes auto-stabilisants pour la construction d'arbres couvrants sous-contraintes. Nous avons notamment détaillé le problème de l'arbre couvrant de diamètre minimum, notion qui est au cœur de notre première contribution. En effet, cette approche est naturelle si l'on veut optimiser le délai de communication entre toutes paires de nœuds puisque celui-ci est limité par le diamètre de l'arbre couvrant, qui est minimal dans le cas d'un MDiamST. C'est pour cela que dans le prochain chapitre, nous allons présenter un algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum sous un démon distribué non équitable.

# Chapitre 3

# Arbre couvrant de diamètre minimum

## 3.1 Introduction

Dans le contexte des réseaux, une tâche cruciale est de maintenir l'efficacité des communications entre les nœuds. Une approche classique pour résoudre ce problème est la construction d'un arbre couvrant le réseau, offrant ainsi un et un unique chemin entre toute paire de nœuds. Il existe différentes sortes d'arbres couvrants selon la métrique que l'on veut optimiser (nombre de sauts à la racine, délai, débit minimum, ...). Le diamètre d'un réseau est l'une des métriques réseau les plus fondamentales. Etre capable de calculer le diamètre est un problème important dans l'analyse de grands réseaux. De plus, ce paramètre a de nombreuses applications pratiques importantes dans des réseaux réels. En conséquence, il est naturel d'étudier ce problème dans un réseau réparti, plus particulièrement dans un réseau réparti tolérant aux fautes transitoires. C'est pour cela que dans ce travail, nous nous concentrons sur le problème de la construction d'un arbre couvrant de diamètre minimum (MDiamST). Le MDiamST est une approche naturelle si l'on veut optimiser le délai de communication entre toutes paires de nœuds dans un réseau, puisque la distance entre une paire de nœuds est limitée par le diamètre de l'arbre qui est minimal dans le cas du MDiamST.

Le problème du MDiamST est étroitement lié au calcul des centres d'un réseau [HT95]. En effet, un centre est un nœud qui minimise son excentricité, autrement dit sa distance maximale à tout les autres nœuds du réseau. Un MDiamST est donc un arbre couvrant BFS enraciné à un centre du réseau. Comme il existe de nombreuses solutions auto-stabilisantes pour calculer l'arbre couvrant BFS, nous nous concentrons dans ce qui suit sur la partie la plus difficile du problème du MDiamST : le problème de calcul du ou des centres du réseau.

Dans cet travail, nous répondons positivement à la question ouverte suivante : est-il possible de calculer les centres d'un réseau d'une manière auto-stabilisante en utilisant uniquement une mémoire de $O(\log n)$ bits par nœud ? Pour cela, nous proposons un nouvel algorithme auto-stabilisant déterministe pour la construction d'un arbre couvrant de diamètre minimum dans un modèle à états. Notre protocole a les caractéristiques intéressantes suivantes :

Premièrement, il est le premier algorithme pour ce problème qui fonctionne sous un

démon distribué non équitable (voir description dans le chapitre 2). En d'autres termes, aucune restriction n'est faite sur le comportement asynchrone du réseau.

Deuxièmement, il améliore la mémoire utilisée par chaque nœud d'un facteur $n$ (où $n$ est le nombre de nœuds du réseau). Notez que ce gain en mémoire ne se fait pas au prix de la performance en temps qui reste polynomiale en nombre de rondes.

**Organisation du chapitre** Ce chapitre est organisé comme suit. En premier lieu, nous présenterons le problème du calcul de l'arbre couvrant de diamètre minimum ainsi qu'un état de l'art relatif à ce problème. Dans la section 3.3.1, nous donnerons le modèle utilisé dans ce travail. Ensuite, dans la section 3.4 nous décrirons notre algorithme. La section 3.5 est dédiée à la preuve de notre algorithme. Enfin, dans la section 3.6, nous discuterons des questions ouvertes sur le problème de l'arbre couvrant de diamètre minimum.

## 3.2 Problème de l'arbre couvrant de diamètre minimum

Le problème de l'arbre couvrant de diamètre minimum (nous rappelons qu'il est noté *MDaimST* pour Minimum Diameter Spanning Tree) est étroitement lié à la détermination des centres du réseau [HT95]. En effet, un centre est un nœud qui minimise son excentricité (autrement dit, sa distance maximale à tout autre nœud du réseau). De plus, il est bien connu qu'un arbre couvrant BFS enraciné en un centre est un MDaimST (voir description BFS et MDiamST dans le chapitre 2). Comme il existe de nombreuses solutions auto-stabilisantes pour la construction d'un BFS, nous nous concentrons dans ce travail sur la partie la plus difficile du problème du calcul de l'arbre couvrant de diamètre minimum, c'est à dire le problème du calcul du centre d'un graphe.

### 3.2.1 Etat de l'art

Une façon naturelle de calculer l'excentricité des nœuds d'un réseau distribué (et par la suite, déterminer ses centres) est de résoudre d'abord le problème du plus court chemin entre toutes paires de nœuds (voir chapitre 1 pour la description du APSP). Une fois qu'un nœud a déterminé les distances à tout les autres nœuds présent dans le réseau, il peut calculer son excentricité. Quand tout les nœuds ont calculé leurs excentricité, il devient facile de trouver les centres du réseau. Le problème APSP a été largement étudié sous diverses hypothèses. Par exemple, [HW12] fournit une bonne étude sur les solutions distribuées récentes pour le calcul du APSP. Dans le même article, les auteurs présentent une solution quasi optimale dans des environnements synchrones. Il existe aussi des résultats d'approximation pour ce problème : [PRT12, RW13], mais ils ne relèvent pas du champ d'application de ce travail puisque nous nous concentrons sur des algorithmes exacts. En conclusion, l'approche APSP est interessante car elle permet d'utiliser des solutions bien connues, mais elle est gourmande en espace mémoire par nœud puisque elle nécessite $O(n \log n)$ bits par nœud.

En revanche, seulement quelques travaux ont porté directement sur le calcul des centres d'un réseau distribué tout en essayant de réduire les besoins en espace comme nous le faisons dans ce travail. Dans un environnement synchrone et sans fautes, nous pouvons citer [KRS84] qui présente le premier algorithme de calcul des centres d'un réseau distribué. Dans un cadre auto-stabilisant, certaines œuvres [AS97, BGKP99, DL13] décrivent des solutions spécifiques à des topologies arborescentes. Le travail le plus lié au nôtre est celui

de Butelle, Lavault et Bui [BLB95]. Le protocole auto-stabilisant distribué proposé dans ce dernier ne fait aucune hypothèse sur la topologie sous-jacente du réseau et fonctionne dans des environnements asynchrones. Son principal inconvénient réside dans sa complexité spatiale de $O(n \log n)$ bits par nœud, qui est équivalent à celui des solutions à base du APSP (voir le chapitre 2 pour une description détaillée de cette solution).

Au meilleur de notre connaissance, la question de savoir s'il est possible de calculer les centres de tout réseau distribué d'une manière auto-stabilisante en utilisant seulement une mémoire sub-linéaire par nœud est toujours ouvert. Notre contribution principale est de répondre positivement à cette question en fournissant un nouvel algorithme auto-stabilisant déterministe qui ne nécessite que $O(\log n)$ bits par nœud, ce qui améliore les résultats existants par un facteur $n$. De plus, notre algorithme est adapté à tout environnement asynchrone puisque nous ne faisons aucune hypothèse sur le démon (ou adversaire) et a un temps de convergence de $O(n^2)$ rondes (qui est comparable avec les solutions existantes [BLB95]).

## 3.3 Algorithme auto-stabilisant pour la construction d'arbre couvrant de diamètre minimum

Dans cette section, nous présentons notre algorithme auto-stabilisant pour le calcul du centre d'un réseau distribué nommé $\mathcal{SSCC}$ (pour $\mathcal{S}$elf $\mathcal{S}$tabilizing $\mathcal{C}$enters $\mathcal{C}$omputation). Nous avons organisé cette section comme suit. Tout d'abord, nous donnons un aperçu général de notre algorithme dans la section 3.3.2. Ensuite, les sections 3.4.1, 3.4.2, 3.4.3, et 3.4.4 sont dédiées à la présentation détaillée de chaque couche de notre algorithme : une couche pour l'élection d'un nœud, une couche pour la circulation du jeton, une couche pour le calcul de l'excentricité et une dernière couche pour le calcul du centre.

### 3.3.1 Modèle

Le réseau est modélisé par un graphe non orienté connexe $G = (V, E)$ où $V$ est l'ensemble des nœuds du réseau et $E$ l'ensemble des liens de communications. Nous considérons des réseaux *non anonyme*. Autrement dit, il existe un identifiant $id_v$ unique pour chaque nœud $v$ pris dans l'ensemble $[0, n^c]$ pour une certaine constante $c$.

Nous considérons le *modèle à états* classique (voir [Dol00]) où chaque nœud a un ensemble de variables stockées dans un registre. Un nœud $v$ peut lire ses propres variables et celles de ses voisins mais ne peut écrire que dans ses propres variables. L'état d'un nœud est défini par la valeur courante de ses variables. L'état d'un réseau (*Configuration*) est le produit des états de tout les nœuds du réseau (voir chapitre 2 ).

L'asynchronisme du réseau est modélisé par un *démon*, qui choisit à chaque étape, le sous-ensemble de nœuds qui sont autorisés à exécuter une de leurs règles pendant cette étape. Notre algorithme est uniforme, il s'exécute dans un environnement totalement asynchrone, c'est à dire sous un démon distribué non équitable (voir la chapitre 2). Plus précisément, soit $A$ l'ensemble des nœuds activables, et $v \in A$ si $A \setminus \{v\} \neq \emptyset$ à chaque étape de l'algorithme, le démon peut choisir de ne jamais activer $v$.

### 3.3.2 Vue d'ensemble de l'algorithme

Nous devons calculer l'excentricité de chacun des nœuds du réseau afin de déterminer le ou les centres du réseau. Le centre du réseau (s'il y en a plusieurs, celui avec l'identifiant minimum est sélectionné) deviendra la racine de l'arbre couvrant de diamètre minimum (MDiamST). Dans l'optique de minimiser l'espace mémoire, les nœuds ne peuvent pas calculer simultanément leurs excentricités, car dans ce cas, l'espace mémoire nécéssaire serait le même que pour le calcul du plus court chemin entre toutes paires de nœuds (APSP), c'est-à-dire $O(n \log n)$. L'idée principale de notre algorithme est basée sur la remarque suivante : le calcul de l'excentricité d'un nœud peut être fait par l'intermédiaire d'un BFS enraciné à ce nœud. Puisque le calcul d'un BFS peut être effectué en utilisant $O(\log n)$ bits de mémoire, il suffit donc d'organiser de façon intelligente les calculs des excentricités afin de maintenir une occupation mémoire de $O(\log n)$ bits par nœud.

Notre algorithme utilise plusieurs couches, chacune d'elle exécutant une tâche spécifique. Avant de présenter plus en détail notre algorithme, en voici un rapide survol. La première couche est consacrée à la construction d'un *arbre couvrant enraciné*, appelé `Backbone`. La deuxième couche maintient *la circulation d'un jeton* sur le `Backbone`. La troisième couche est dédiée aux *calculs des excentricités* : un nœud qui possède le jeton calcule son excentricité avant de passer le jeton à son voisin sur le `Backbone`, qui en fait de même et ainsi de suite. La dernière couche est consacrée au *calcul du centre* : le `Backbone` collecte les excentricités des feuilles vers la racine. La racine du `Backbone` calcule le centre du graphe, et diffuse l'identifiant de ce centre par l'intermédiaire du `Backbone`.

Pour converger vers un MDiamST, nos couches doivent avoir différentes priorités. En effet, la construction du `Backbone` est la couche ayant la plus haute priorité, le reste de notre algorithme ne pouvant s'exécuter correctement si notre `Backbone` n'est pas correct. Il en va de même pour la circulation de jeton, notre algorithme doit s'assurer de l'unicité du jeton pour effectuer un calcul correct des excentricités. Enfin, le calcul des excentricités et le calcul du centre du graphe peuvent être fait de manière concurrente.

Les principales difficultés rencontrées pour implémenter cette approche sont l'utilisation de mêmes variables pour le calcul des différentes excentricités et l'organisation des différentes couches afin qu'elles fonctionnent avec un démon totalement asynchrone. Nous allons maintenant détailler les différentes couches, pour la première et la deuxième couche nous utilisons des résultats existants, le cœur de notre travail est donc l'organisation entre ces couches et les deux dernières couches.

La première couche est consacrée à la construction d'un *arbre couvrant enraciné*. Nous considérons dans ce travail un réseau uniforme dans lequel tout les nœuds exécutent le même algorithme auto-stabilisant, il n'y a donc à priori aucun nœud qui peut jouer de rôle particulier et plus précisément le rôle de racine. La première tâche est donc l'élection d'une racine. Il nous faut un algorithme qui fonctionne sous un démon distribué non équitable, qui utilise $O(\log n)$ bits de mémoire par nœud, et qui se stabilise en $O(n)$ rondes. Il existe deux algorithmes dans la littérature correspondants à ces critères, celui de Datta, Larmore et Vemula [DLV11] et celui de Devismes et Johnen [DJ15]. Le lecteur intéressé peut se reporter à ces deux articles. Comme l'algorithme est utilisé comme une boîte noire, nous n'avons pas besoin de le présenter formellement dans ce document. En effet, les auteurs de ces articles ont conçu un algorithme auto-stabilisant pour construire un arbre BFS enraciné au nœud possédant l'identifiant minimum. Ces algorithmes s'auto-stabilisent sous un démon distribué non equitable, ils utilisent $O(\log n)$ bits de mémoire par nœud et ils convergent en $O(n)$ rondes. Dans le reste de l'article, nous appelons l'arbre BFS construit

par cette première couche le `Backbone` du réseau.

La seconde couche est une circulation de jeton sur le `Backbone`. Parmi les algorithmes auto-stabilisants existants pour la circulation de jeton, nous avons choisi d'adapter l'algorithme de Petit et Villain [PV99]. Le but de cette circulation de jeton est de synchroniser le multiplexage temporel des variables de la troisième couche de notre algorithme qui calcule l'excentricité de chaque nœud. En effet, afin de réduire la complexité en espace de notre algorithme à $O(\log n)$ bits par nœud, tout les nœuds calculent leurs excentricités en utilisant les mêmes variables mais les uns après les autres. Pour éviter les conflits, nous gérons cette exclusion mutuelle par une circulation symbolique. Plus précisément, on distingue, pour chaque nœud, la circulation de jeton ascendante (dans laquelle le nœud prend le jeton de l'un de ses enfants dans le `Backbone`) et la circulation descendante (dans laquelle le nœud reprend le jeton de l'un des ses enfants dans le `Backbone`).

La composition entre la couche de construction du `Backbone` et la couche de circulation du jeton de notre algorithme doit résister à la non équité du démon. En effet, nous devons nous assurer que le démon ne peut pas choisir exclusivement les nœuds qui souhaitent exécuter la circulation de jeton (rappelons que nous supposons que la construction du `Backbone` a la priorité sur la circulation de jeton) car cela pourrait empêcher le `Backbone` d'être construit. Pour répondre cette question, nous avons choisi d'ordonner différentes règles afin de forcer la construction du `Backbone`.

La troisième couche de notre algorithme est dédiée aux *calculs des excentricités*. Nous différencions la circulation du jeton descendante de celle ascendante. En montée, le jeton n'effectue aucune tâche supplémentaire. Quand un nœud $v$ reçoit le jeton **en descente** il lance le calcul de son excentricité de la façon suivante. Quand $v$ reçoit le jeton en descente, il commence une construction auto-stabilisante d'un arbre BFS enraciné en lui-même. Quand la construction du BFS de $v$ est accomplie, la profondeur maximale du BFS est calculée des feuilles vers $v$, cette profondeur maximale est en fait l'excentricité de $v$. Une fois que le nœud $v$ a recueilli son excentricité, il libère le jeton pour le nœud suivant dans le `Backbone`. Cette troisième couche pourrait être un résultat en lui même, car le calcul de l'excentricité d'un nœud à l'aide d'un BFS (mémoire $O(\log n)$) pose plusieurs défis dû à l'asynchronisme. En effet, certaines branches du BFS peuvent commencer à calculer leur profondeur avec une mauvaise estimation de leurs distances à $v$. Pour résoudre ce problème, nous avons mis en place une priorité sur le calcul du BFS par rapport au calcul de la profondeur, tout changement de valeur de la distance dans le BFS entraînant une purge du calcul de la profondeur.

Enfin, la quatrième couche est dédiée au *calcul du centre*. L'excentricité de chaque nœud est collectée à chaque instant des feuilles vers la racine du `Backbone`. Ensuite, la racine propage cette excentricité minimum à tout les nœuds le long du `Backbone`. Le nœud avec l'excentricité minimum et l'identifiant minimum devient le centre du réseau. Ce nœud construit un BFS, le résultat est un arbre couvrant de diamètre minimum. Notre solution évite une configuration arbitraire de départ construisant plusieurs BFS à partir de plusieurs nœuds se considérant de façon abusive comme centre, en dédiant uniquement trois variables à la construction du MDiamST (racine, parent, distance), évitant par la même l'utilisation excessive de mémoire.

Nous prouvons que chaque couche de cet algorithme empilée est auto-stabilisante et que leur composition s'auto-stabilise à un arbre couvrant de diamètre minimum sous un démon distribué non équitable au bout de $O(n^2)$ rondes. Comme chaque couche de notre algorithme a besoin d'au plus $O(\log n)$ bits par nœud, nous obtenons la complexité en espace

désiré. Notons que notre algorithme auto-stabilisant pour la construction d'un MDiamST est non-silencieux car l'algorithme de circulation de jeton transporte une information qui évolue continuellement même après la convergence de l'algorithme (voir le chapitre 2 pour la définition d'un algorithme silencieux).

## 3.4 Description détaillée de l'algorithme

Dans cette section, nous allons détailler le fonctionnement des couches de notre algorithme.

### 3.4.1 Election d'une racine élue et construction d'un arbre couvrant

La première couche de notre algorithme exécute un algorithme auto-stabilisant d'élection basé sur la construction d'un arbre couvrant. Il nous faut un algorithme qui fonctionne sous un démon distribué non équitable, qui utilise $O(\log n)$ bits de mémoire par nœud, et qui se stabilise en $O(n)$ rondes. Il existe deux algorithmes dans la littérature correspondants à ces critères, celui de Datta, Larmore et Vemula [DLV11]et celui de Devismes et Johnen [DJ15].

Il est important de rappeler que `Backbone` est l'arbre BFS construit par cette couche de notre algorithme. Pour le reste de la présentation, nous notons le parent de tout nœud $v$ dans le `Backbone` par $\mathtt{p}_v$, l'ensemble des enfants de $v$ dans le `Backbone` par child($v$) et l'ensemble des voisins de $v$ dans le `Backbone` par $\mathrm{N}_{\mathtt{Backbone}}(v)$. Autrement dit, si le `Backbone` est défini par le 1-facteur $\{(v, \mathtt{p}_v), v \in V\}$, alors nous avons $\mathrm{child}(v) = \{u \in V : \mathtt{p}_u = v\}$ et $\mathrm{N}_{\mathtt{Backbone}}(v) = \mathrm{child}(v) \cup \{\mathtt{p}_v\}$. Nous définissons également le prédicat BRoot($v$) sur les variables de cette couche. Ce prédicat est vrai si et seulement si le nœud $v$ est la racine du `Backbone`.

Il est important de noter que la construction du `Backbone` a une priorité plus élevée que les autres couches de notre algorithme (qui sont : la circulation de jeton, le calcul de l'excentricité et la détermination des centres). Dans le contexte des algorithmes auto-stabilisants uniformes et silencieux, les trois variables nécessaires et suffisantes pour prouver la cohérence d'un arbre couvrant [BF15] sont :

— L'identifiant de la racine (qui doit être le même pour tout les nœuds).

— Le pointeur parent (qui doit être un voisin du nœud, sauf pour la racine qui n'a pas de parent).

— La distance du nœud (qui doit être la distance de son parent plus un).

En d'autres termes, si un nœud $v$ a un voisin avec une racine différente dans le `Backbone` ou $v$ une distance incohérente dans le `Backbone` par rapport à son parent, $v$ ne peut pas exécuter une règle liée à une autre couche. Cette priorité est nécessaire pour que notre algorithme fonctionne sous un démon non équitable.

### 3.4.2 Circulation du jeton

La deuxième couche de notre algorithme est une légère adaptation d'un algorithme de circulation de jeton auto-stabilisant proposé par petit et villain [PV99]. Le jeton circule infiniment sur le `Backbone` dans un ordre DFS. Cet algorithme fonctionne sous un démon distribué non équitable, il utilise $O(\log n)$ bits de mémoire par nœud et converge en $O(n)$

rondes. Nous rappelons que nous ne modifions pas la circulation ascendante du jeton, mais que nous retardons la circulation descendante par le calcul de l'excentricité du nœud possédant le jeton.

Cet algorithme utilise une seule variable pour chaque nœud $v$ : la variable $\mathsf{next}_v \in \{\perp, \text{done}, N_v\}$. Cette variable stocke l'état du nœud par rapport à la circulation du jeton courant. La valeur $\perp$ signifie que le nœud n'a pas déjà été visité par le jeton. Lorsque $\mathsf{next}_v$ pointe vers un enfant de $v$ dans le `Backbone`, cela signifie que $v$ a été visité par le jeton et que $v$ a envoyé le jeton à son enfant pointé par $\mathsf{next}_v$. La valeur done signifie que le nœud $v$ et tous ses enfants dans le `Backbone` ont déjà été visités par le jeton. En d'autres termes, le jeton est détenu par le premier nœud $v$ avec $\mathsf{next}_v = \perp$ le long du chemin issu de la racine du `Backbone` suivant les variables `next` (pointeurs).

La Figure 3.1 illustre la circulation du jeton. Nous définissons un ordre total pour $\succ$ sur les valeurs de la variable `next` en étendant l'ordre naturel $>$ sur les identifiants avec l'hypothèse suivante : pour tout nœud $v$, nous avons $\text{done} \succ \text{ID}_v \succ \perp$.

La présentation formelle de cette couche est fournie par l'algorithme 1, la fonction et les prédicats utilisés par cet algorithme sont dans la suite du document. Cet algorithme se compose de deux règles. La première, $\mathbb{R}_{\texttt{ErToken}}$ assure la convergence vers un jeton unique, tandis que la seconde, $\mathbb{R}_{\texttt{Backward}}$ assure la circulation descendante du jeton. La règle de circulation ascendante est pris en charge par la couche suivante de notre protocole (voir ci-dessous pour plus d'explications sur la relation entre ces deux couches).

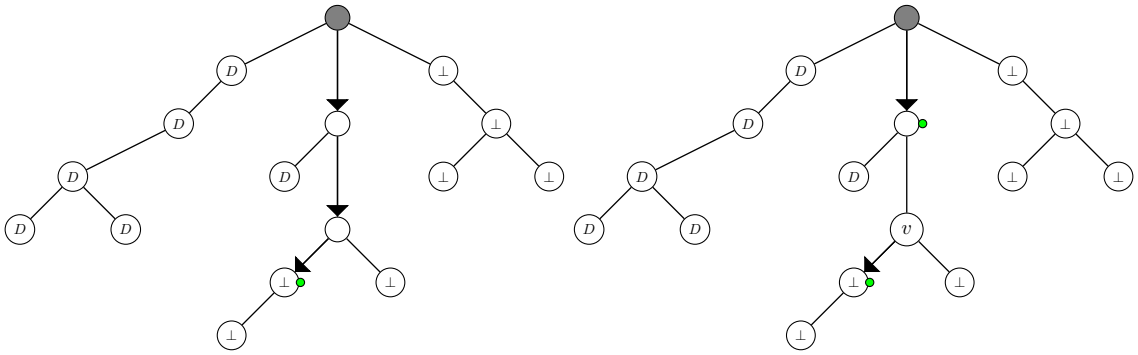
**Algorithm 1:** Circulation du jeton pour un nœud $v$

$\mathbb{R}_{\texttt{ErToken}}$ :$\text{ErValues}(v) \vee \big(\neg\text{BRoot}(v) \wedge (\mathsf{next}_{\mathsf{p}_v} \neq v) \wedge (\mathsf{next}_v \in \text{child}(v) \cup \{\text{done}\})\big) \rightarrow \mathsf{next}_v := \perp$;

$\mathbb{R}_{\texttt{Backward}}$:$\neg\text{ErValues}(v) \wedge (\text{BackNd}(v) \vee \text{BackR}(v)) \wedge (\mathsf{R}_{\mathsf{BFS}_{\text{FNext}(v)}} \neq \text{ID}(\text{FNext}(v))) \rightarrow \mathsf{next}_v := \text{FNext}(v)$;

Nous avons modifié l'algorithme original de [PV99] de la manière suivante. Quand un nœud $v$ reçoit le jeton en descente, ce dernier est retenu par $v$ jusqu'à ce que la troisième couche de notre protocole calcule l'excentricité de $v$. Cela se fait par la construction d'un arbre BFS enraciné à $v$ et par la collecte de la distance maximale entre $v$ et tout les autres nœuds du réseau (voir la section 3.4.3). Ceci est la raison pour laquelle la circulation descendante du jeton n'est pas effectuée par une règle de l'algorithme 1, mais par la règle $\mathbb{R}_{\texttt{EndBFS}}$ dans l'algorithme 2 (qui décrit la troisième couche de notre algorithme). La communication entre ces deux couches sur l'état du jeton est effectuée en utilisant le prédicat TokenD($v$). Lorsque la couche de circulation du jeton donne le jeton au nœud $v$, ce prédicat devient vrai, ce qui permet à la couche calculant l'excentricité de commencer. Une fois l'excentricité du nœud $v$ calculée, le nœud $v$ met à jour $next_v$ (voir $\mathbb{R}_{\texttt{EndBFS}}$ dans l'algorithme 2) ce qui a pour effet de déclencher à nouveau la circulation descendante du jeton (exactement comme dans l'algorithme original de [PV99]).

Nous avons également légèrement modifié la circulation du jeton ascendante afin d'assurer la convergence de la couche de calcul de l'excentricité. Si le nœud $v$ veut envoyer le jeton en circulation ascendante à un voisin $u$, $v$ doit attendre dans le cas où $u$ est lui aussi en train de calculer son excentricité (cette situation est possible si $u$ croit à tort avoir le jeton). Nous effectuons cette attente grâce à la variable $\mathsf{R}_{\mathsf{BFS}_u}$ dédiée à la construction du BFS (voir la section 3.4.3). Cette variable stocke l'identifiant de la racine du BFS en cours de construction. Ensuite, $v$ redémarre la circulation descendante. Si $u$ croit à tort avoir le




(a) Configuration normale (b) Erreur detecter en nœud $v$

FIGURE 3.1 – Circulation du jeton

jeton, il est capable de détecter localement cette erreur ainsi que de la corriger, et ceci en temps fini (voir la section 3.4.3). Nous assurons donc que le jeton ne soit jamais infiniment bloqué par $v$.

La composition entre la couche de construction du `Backbone` et la couche de circulation du jeton de notre algorithme doit résister à l'iniquité du démon. En effet, nous devons nous assurer que le démon ne peut pas choisir exclusivement les nœuds activables uniquement pour les règles de circulation de jeton (rappelons que que la construction du `Backbone` est prioritaire par rapport à la circulation de jeton) car cela pourrait conduire à un arrêt de la construction du `Backbone`. Pour faire face à ce problème, nous avons choisi de bloquer la circulation du jeton au nœud $v$ si $v$ a un voisin qui ne fait pas partie du `Backbone` ou si $v$ détecte une incohérence entre les distances dans le `Backbone` ( voir la définition et l'utilisation de prédicat ErValues).

Ainsi, avant la stabilisation du `Backbone`, la structure couvrante induite par des variables $\mathsf{p}_v$ pour tout $v \in V$ peut être composée uniquement de sous-arbres ou de cycles. Nous devons donc en premier assurer la construction du `Backbone`. En effet, le démon ne peut pas activer à l'infini les règles de circulation du jeton des nœuds dans un sous-arbre donné, car dans ce cas au moins un des nœuds $v$ de ce sous-arbre a un voisin qui ne fait pas partie du même sous-arbre, en consequence $v$ bloquera le jeton. De même, le démon ne peut pas activer à l'infini les règles de circulation du jeton dans un cycle donné, car dans un cycle il y a au moins un nœud $v$ qui va détecter une incohérence de distance dans le `Backbone`. Dans ce cas $v$ bloquera le jeton ce qui aura comme effet de redonner la priorité à la construction du `Backbone`.

### Description détaillée de la circulation du jeton

Nous commençons par la description détaillée de la règle $\mathbb{R}_{\texttt{ErToken}}$. Nous rappelons que la règle $\mathbb{R}_{\texttt{ErToken}}$ est utilisée pour permettre la convergence vers un jeton unique.

$$\mathbb{R}_{\texttt{ErToken}} : \text{ErValues}(v) \vee \big(\neg\text{BRoot}(v) \wedge (\mathsf{next}_{\mathsf{p}_v} \neq v) \wedge (\mathsf{next}_v \neq \bot) \longrightarrow \mathsf{next}_v := \bot; \tag{3.1}$$

Nous rappelons que pour la variable $\mathsf{next}_v$ la valeur $\bot$ est inférieure à touts les identi-

fiants et à la valeur done.

Cette règle est dédiée à la détection d'une erreur par le nœud $v$. Quand une erreur est détectée par $v$, il supprime sa variable $\mathsf{next}_v$ ($\mathsf{next}_v = \perp$). Le prédicat ErValues($v$) est dédié à la détection d'une erreur par le nœud $v$. La valeur $\perp$ signifie que le nœud n'a pas déjà été visité par le jeton ou qu'il détient le jeton. Lorsque $\mathsf{next}_v$ pointe vers un enfant de $v$ dans le `Backbone`, cela signifie que $v$ a été visité par le jeton et que $v$ a envoyé le jeton à son enfant pointé par $\mathsf{next}_v$. La valeur done signifie que le nœud et tous ses enfants dans le `Backbone` ont déjà été visités par le jeton. Par conséquent, si le jeton est différent de l'une de ces variables, une erreur est alors détectée.

$$\mathrm{ErValues}(v) \equiv \mathsf{next}_v \notin \mathrm{child}(v) \cup \{\perp, \mathrm{done}\} \tag{3.2}$$

La deuxième partie de la garde de la règle $\mathbb{R}_{\texttt{ErToken}}$ est dédiée à détecter une autre incohérence des variables dédiées à la circulation du jeton. En effet, si un nœud $v$ n'est pas la racine du `Backbone` et que ses variables $\mathsf{next}_v$ pointent envers l'un de ses enfants, cela signifie que le jeton est stocké par l'un de ses descendants, donc son parent doit pointer vers lui (voir la Figure 3.1). Finalement, cette règle détecte aussi le cas où le jeton explore déjà l'ensemble du sous-arbre enraciné au parent du nœud $v$ et définit ensuite sa variable next à $\perp$ pour préparer la prochaine circulation du jeton. En effet, dans ce cas, nous avons : $\mathsf{next}_v = \mathrm{done}$ et $\mathsf{next}_{\mathsf{p}_v} = \mathrm{done}$ et l'exécution de la règle ré-initialise $\mathsf{next}_v$ à $\perp$.

La règle $\mathbb{R}_{\texttt{Backward}}$ effectue la circulation ascendante du jeton. Nous rappelons que la règle de circulation descendante est laissée à la couche suivante de notre algorithme.

$$\begin{aligned}\mathbb{R}_{\texttt{Backward}}\colon \neg\mathrm{ErValues}(v) \wedge (\mathrm{BackNd}(v) \vee \mathrm{BackR}(v)) \wedge (\mathsf{R}_{\mathrm{BFS}_{\mathrm{FNext}(v)}} \neq \mathrm{ID}(\mathrm{FNext}(v))) \\ \longrightarrow \mathsf{next}_v := \mathrm{FNext}(v);\end{aligned} \tag{3.3}$$

La règle $\mathbb{R}_{\texttt{Backward}}$ est activée si et seulement si le nœud $v$ n'est pas en erreur (voir prédicat $\neg$ErValues($v$)). Afin de pouvoir expliquer cette règle, nous devons d'abord expliquer la fonction FNext($v$). Cette fonction retourne l'identifiant du prochain voisin de $v$ auquel le jeton doit être envoyé (ordre lexicographique sur les identifiants) ou done si tout le sous-arbre enraciné au nœud $v$ a déjà été exploré. Cette fonction permet également à la racine de l'arbre de lancer la re-initialisation des variables next à $\perp$ quand il détecte localement la fin d'une circulation de jeton.

$$\mathrm{FNext}(v) = \begin{cases} u & \text{si } \exists u \in \mathrm{child}(v), \mathrm{ID}_u = \min\{\mathrm{ID}_w \mid w \in \mathrm{child}(v) \wedge (\mathrm{ID}_w \succ \mathsf{next}_v)\} \\ \perp & \text{si } \mathrm{BRoot}(v) \wedge \mathsf{next}_v \in \mathrm{child}(v) \wedge (\forall u \in \mathrm{child}(v), \mathsf{next}_v \succ \mathrm{ID}_u \vee \mathsf{next}_v = \mathrm{ID}_u) \\ \mathrm{done} & \text{sinon} \end{cases} \tag{3.4}$$

Les prédicats BackR($v$) et BackNd($v$) utilisés dans la règle $\mathbb{R}_{\texttt{Backward}}$ sont essentiellement utilisés pour détecter le moment où le jeton termine sa circulation dans le sous-arbre enraciné au niveau du nœud actuellement pointé par $\mathsf{next}_v$ (indiqué par la valeur done).

$$\text{BackR}(v) \equiv \text{BRoot}(v) \wedge (\mathsf{next}_v = u) \wedge (\mathsf{next}_u = \text{done}) \wedge \text{PermR}(v) \tag{3.5}$$

$$\begin{aligned}\text{BackNd}(v) \equiv \neg\text{BRoot}(v) \wedge (\mathsf{next}_{\mathsf{p}_v} = v) \wedge (\mathsf{next}_v \in \text{child}(v)) \\ \wedge(\mathsf{next}_{next_v} = \text{done}) \wedge \text{PermNd}(v)\end{aligned} \tag{3.6}$$

Ces prédicats font usage respectivement de PermR($v$) et PermNd($v$). Ces derniers sont utilisés pour faire en sorte que le nœud $u$ retourné par l'application de la fonction FNext($v$) satisfait $\mathsf{next}_u = \perp$ (afin d'éviter l'envoi du jeton sur un sous-arbre dans un état illégal).

$$\text{PermR}(v) \equiv (\text{FNext}(v) = \perp) \vee (\mathsf{next}_{\text{FNext}(v)} = \perp) \tag{3.7}$$

$$\text{PermNd}(v) \equiv (\text{FNext}(v) = \text{done}) \vee (\mathsf{next}_{\text{FNext}(v)} = \perp) \tag{3.8}$$

La communication entre la couche circulation de jeton et la couche du calcul de l'excentricité est effectuée en utilisant le prédicat TokenD($v$). Lorsque la couche de circulation du jeton donne le jeton au nœud $v$, ce prédicat devient vrai, ce qui permet à la couche du calcul de l'excentricité de commencer. Une fois que l'excentricité du nœud $v$ est calculée, cette dernière met à jour $next_v$ qui effectue la circulation du jeton descendante. Et ceci exactement comme dans l'algorithme original de [PV99].

$$\begin{aligned}\text{TokenD}(v) \equiv (\mathsf{next}_v = \perp) \wedge \quad & \big((\text{BRoot}(v) \wedge \text{PermR}(v)) \vee \\ & (\neg\text{BRoot}(v) \wedge (\mathsf{next}_{\mathsf{p}_v} = v) \wedge \text{PermNd}(v))\big)\end{aligned} \tag{3.9}$$

### 3.4.3 Calcul de l'excentricité

La troisième couche de notre algorithme est consacrée au calcul de l'excentricité de chaque nœud. Rappelons que la circulation du jeton descendante effectuée par la seconde couche assure qu'au plus un nœud calcule son excentricité à la fois, ce qui nous permet de réutiliser les mêmes variables. Brièvement, l'excentricité de chaque nœud est calculée comme suit. Premièrement, le nœud commence la construction d'un arbre couvrant BFS enraciné en lui-même. Une fois cela fait, nous recueillons la distance maximale dans cet arbre (à savoir, l'excentricité de sa racine) à partir des feuilles. Ensuite, le nœud obtient son excentricité et libère le jeton. Cet algorithme fonctionne sous le démon distribué non équitable, il utilise $O(\log n)$ bits de mémoire par nœud, et se stabilise en $O(n)$ rondes (pour le calcul de l'excentricité d'un nœud uniquement).

Pour la clarté de la présentation, notons $r$ un nœud qui obtient le jeton à un moment donné. Il est important de noter qu'à partir de ce moment et jusqu'à la libération du jeton par ce nœud, le prédicat TokenD(r) restera vrai. Notre algorithme commence la construction de $\texttt{BFS}(r)$ (l'arbre couvrant enraciné à $r$). La première étape consiste à informer tout les nœuds de l'identifiant du nœud possédant le jeton, autrement dit du nœud $r$, afin de synchroniser leurs algorithmes de calcul d'excentricité. Nous faisons cela en diffusant l'identifiant de $r$ le long du `Backbone`. Nous réalisons cette diffusion en réorientant temporairement le `Backbone` vers $r$. Ensuite, nous utilisons une construction BFS classique empruntée de [HC92] qui consiste, pour chaque nœud, de choisir comme parent dans l'arbre le nœud parmi ses voisins qui propose la plus petite distance à la racine. Evidemment, le nœud met ensuite à jour sa propre distance pour être cohérent avec celle de son nouveau parent.

La partie délicate est de collecter la distance maximale à la racine après la stabilisation du $\texttt{BFS}(r)$ (et pas avant). Comme nous l'avons dit précédemment, cette collecte est faite par une vague à partir des feuilles jusqu'à la racine de $\texttt{BFS}(r)$. Chaque feuille du $\texttt{BFS}(r)$ propage à son parent sa propre valeur d'excentricité, tandis que les nœuds internes prennent pour excentricité le maximum entre les valeurs d'excentricités de leurs enfants dans $\texttt{BFS}(r)$.

En raison de l'asynchronisme du réseau, certaines difficultés peuvent apparaître lors de ce processus. En effet, si nous recueillons une valeur d'excentricité dans une branche du $\texttt{BFS}(r)$ alors que cette branche n'est pas encore stabilisée (puisque certains nœuds peuvent encore se joindre à elle), nous pouvons obtenir une excentricité erronée pour le nœud $r$. Dans ce cas, $r$ peut libérer le jeton plus tôt que prévu. Pour éviter cela, nous gérons la collecte de la distance maximale de la manière suivante. Lorsqu'un un nœud $v$ change de distance $\texttt{BFS}(r)$, ce nœud nettoie sa variable excentricité. Cela consiste à effacer la valeur actuelle de cette variable et de la remplacer par une valeur spécifique.

Ensuite, tout les nœuds sur le chemin de $\texttt{BFS}(r)$ entre $r$ et $v$ nettoient leurs variables excentricité dans un processus ascendant. En d'autres termes, nous maintenons au moins un chemin dans $\texttt{BFS}(r)$ dans lequel toutes les variables excentricité sont nettoyées jusqu'à ce que $\texttt{BFS}(r)$ soit stabilisé. L'existence de ce chemin nous assure que $r$ n'obtient pas son excentricité et libère le jeton de façon prématurée.

**Description formelle** Nous sommes maintenant en mesure de présenter formellement notre algorithme de calcul d'excentricité. Tout d'abord, nous rappelons que, dans le but de diffuser l'identifiant du nœud $r$ dont l'algorithme calcule actuellement l'excentricité, nous devons réorienter le `Backbone` pour l'enraciner en $r$. Nous appelons cet arbre orienté $\texttt{Backbone}(r)$. A cet effet, nous introduisons la fonction p_next($v$) pour chaque nœud $v$ (Illustration de la variable p_next dans la Figure 3.2 ). Cette fonction retourne l'identifiant du voisin de $v$ qui appartient au chemin du `Backbone` de $v$ à $r$. Plus précisément, p_next est définie comme suit :

$$\text{p\_next}(v) = \begin{cases} \mathsf{p}_v & \text{si TokenD}(v) = false \wedge \mathsf{next}_v \in \{\bot, \text{done}\} \\ \bot & \text{si TokenD}(v) = true \\ \mathsf{next}_v & \text{sinon} \end{cases} \tag{3.10}$$

Nous définissons aussi une fonction qui retourne l'ensemble des enfants de $v$ dans le $\texttt{Backbone}(r)$. C'est à dire, les voisins $u$ de $v$ qui satisfont p_next($u$) = $v$. Notre algorithme utilise les variables suivantes pour chaque nœud $v$ pour construire $\texttt{BFS}(r)$ et calculer l'excentricité de $r$ :

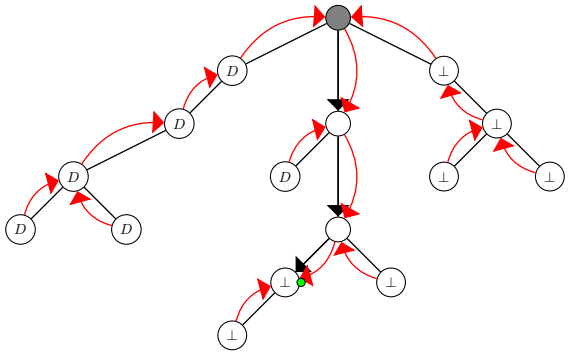



FIGURE 3.2 – Illustration de la fonction p_next.
Les flèches noires représentent la variable next les flèches rouges la variable p_next.

— $\mathsf{Ecc}_v \in \mathbb{N} \cup \{\bot\}$ est l'excentricité du nœud $v$ ;
— $\mathsf{R}_{\mathtt{BFS}v} \in \mathbb{N}$ est l'identifiant de la racine dans $\mathtt{BFS}(r)$ ;
— $\mathsf{P}_{\mathtt{BFS}v} \in \mathbb{N} \cup \{\bot\}$ est l'identifiant du parent de $v$ dans $\mathtt{BFS}(r)$ ;
— $\mathsf{d}_{\mathtt{BFS}v} \in \mathbb{N} \cup \{\bot, \infty\}$ est la distance entre $v$ et la racine dans $\mathtt{BFS}(r)$ ;
— $\mathsf{D}_{\mathtt{BFS}v} \in \mathbb{N} \cup \{\bot, \downarrow, \uparrow\}$ est la distance maximale entre la racine $r$ et la feuille la plus éloignée dans le sous-arbre de $v$ dans $\mathtt{BFS}(r)$ ;

Maintenant, nous pouvons présenter les règles de la troisième couche de notre algorithme. Ces règles font usage de certains prédicats et fonctions qui sont décrites dans la section 3.4.3. Pour la clarté de la présentation, nous avons divisé les règles de notre algorithme en deux ensembles. Le premier (voir Algorithm 2) contient des règles activées pour un nœud qui détient le jeton (c'est-à-dire le nœud $v$ tel que TokenD$(v) = true$) tandis que le second (voir Algorithm 3) décrit les règles d'un nœud qui ne détient pas le jeton.

Nous discutons d'abord des règles activées lorsque le nœud détient le jeton présenté dans l'algorithme 2. Rappelons que ces règles ne sont appliquées que lorsque le nœud reçoit le jeton dans une circulation descendante (voir la section 3.4.2). Une fois que le nœud $r$ a reçu le jeton, son prédicat TokenD$(r)$ devient vrai. Dans cet état, le nœud $r$ peut appliquer seulement trois règles.

**Algorithm 2:** Calcul de l'excentricité d'un nœud $v$ tel que TokenD$(v) = true$

$$\mathbb{R}_{\mathtt{StartBFS}} : \neg\text{RootBFS}(v) \longrightarrow (\mathsf{R}_{\mathtt{BFS}v}, \mathsf{d}_{\mathtt{BFS}v}, \mathsf{D}_{\mathtt{BFS}v}) := (\text{ID}_v, 0, \bot)$$

$$\mathbb{R}_{\mathtt{CleanEcc}} : \text{RootBFS}(v) \wedge \text{SameBFS}(v) \wedge (\text{Down}(v) \vee \text{Bot}(v)) \longrightarrow \mathsf{D}_{\mathtt{BFS}v} = \downarrow \text{ if Down}(v) \longrightarrow \mathsf{D}_{\mathtt{BFS}v} = \bot \text{ if Bot}(v)$$

$$\mathbb{R}_{\mathtt{EndBFS}} : \text{RootBFS}(v) \wedge \text{SameBFS}(v) \wedge \text{ChN}(v) \longrightarrow \mathsf{Ecc}_v := \mathsf{MaxD}(v); \mathsf{next}_v := \text{FNext}(v); \mathsf{MinEUP}_v := \mathsf{MinEcc}(v);$$

La règle $\mathbb{R}_{\mathtt{StartBFS}}$ commence le calcul de $\mathtt{BFS}(r)$ lorsque $r$ prend un état indiquant qu'il est la racine de l'arbre couvrant BFS. La règle $\mathbb{R}_{\mathtt{CleanEcc}}$ nettoie la variable d'excentricité de

$r$ en cas de besoin (par exemple après un faux calcul de l'excentricité du à l'asynchronisme du réseau). Enfin, la règle $\mathbb{R}_{\texttt{EndBFS}}$ est exécutée lorsque la propagation de l'excentricité des feuilles à la racine est terminée. Cette règle calcule l'excentricité de $r$, libère le jeton en mettant à jour la variable $\mathsf{next}_r$ de la couche de circulation du jeton (voir la section 3.4.2) et met à jour une variable pour communiquer la nouvelle excentricité à la couche de determination des centres (voir la section 3.4.4).

Nous nous concentrons maintenant sur les règles activées lorsque les nœuds ne possèdent pas le jeton (Algorithme 3). La première règle $\mathbb{R}_{\texttt{Tree}}$ est consacrée à propager l'identifiant de la racine $r$ de l'arbre couvrant BFS le long du `Backbone`. Cette propagation est possible puisque nous réorientons le `Backbone` (se référer à la définition de p_next ci-dessus). Dans le même temps, la règle $\mathbb{R}_{\texttt{Tree}}$ détecte également quelques erreurs locales. A titre d'exemple, la variable $\mathsf{D}_{\mathsf{BFS}_v}$ d'un nœud (utilisée pour recueillir l'excentricité de $r$) ne doit pas avoir un entier si l'un des enfants (dans $\texttt{BFS}(r)$) de ce nœud n'est pas dans le même cas. La règle $\mathbb{R}_{\texttt{BFS}}$ exécute la construction BFS en elle-même.

Enfin, la règle $\mathbb{R}_{\texttt{Ecc}}$ traite de la phase délicate de collecte de l'excentricité des feuilles à la racine avec le mécanisme de nettoyage expliqué ci-dessus. Ces règles sont mises en œuvre avec l'aide de prédicats et fonctions définis et détaillés dans la suite du chapitre.

**Algorithm 3:** Calcul de l'excentricité d'un nœud $v$ tel que TokenD$(v) = false$

$$\mathbb{R}_{\texttt{Tree}} : \neg\text{GoodBFS}(v) \longrightarrow (\mathsf{R}_{\mathsf{BFS}_v}, \mathsf{P}_{\mathsf{BFS}_v}, \mathsf{d}_{\mathsf{BFS}_v}, \mathsf{D}_{\mathsf{BFS}_v}) := (\mathsf{R}_{\mathsf{BFS}_{\text{p\_next}(v)}}, \perp, \infty, \perp)$$

$$\mathbb{R}_{\texttt{BFS}} : \text{GoodBFS}(v) \wedge \text{SameBFS}(v) \wedge (\text{Best}(v) \neq \perp) \longrightarrow \text{if } \mathsf{D}_{\mathsf{BFS}_v} \in \mathbb{N} \text{ alors } \mathsf{D}_{\mathsf{BFS}_v} :=\uparrow; \text{ sinon } \mathsf{D}_{\mathsf{BFS}_v} :=\downarrow; \longrightarrow (\mathsf{P}_{\mathsf{BFS}_v}, \mathsf{d}_{\mathsf{BFS}_v}) := (\text{Best}(v), \mathsf{d}_{\mathsf{BFS}_{\text{Best}(v)}} + 1);$$

$$\mathbb{R}_{\texttt{Ecc}} : \text{GoodBFS}(v) \wedge \text{SameBFS}(v) \wedge (\forall u \in N_v : \mathsf{d}_{\mathsf{BFS}_u} \neq \infty) \wedge (\text{Best}(v) = \perp) \wedge (\mathsf{D}_{\mathsf{BFS}_v} \neq \text{Dis}(v)) \longrightarrow \mathsf{D}_{\mathsf{BFS}_v} := \text{Dis}(v)$$

### Description détaillée du calcul de l'excentricité

Dans cette section, nous détaillons les règles des algorithmes 2 et 3. La règle $\mathbb{R}_{\texttt{StartBFS}}$ est dédiée au nœud $v$ avec TokenD$(v) = true$. Le nœud $v$ commence le calcul de $\texttt{BFS}(v)$, les variables dédiées au calcul BFS prennent des valeurs correspondantes à la racine. Nous rappelons que, $\mathsf{R}_{\mathsf{BFS}_v}$ est consacrée à identifier la racine dans le BFS courant, $\mathsf{d}_{\mathsf{BFS}_v}$ est la distance de la racine à $v$ dans le BFS actuel et $\mathsf{D}_{\mathsf{BFS}_v}$ est utilisée pour calculer l'excentricité de la racine du BFS courant.

$$\mathbb{R}_{\texttt{StartBFS}} : \neg\text{RootBFS}(v) \longrightarrow (\mathsf{R}_{\mathsf{BFS}_v}, \mathsf{d}_{\mathsf{BFS}_v}, \mathsf{D}_{\mathsf{BFS}_v}) := (\text{ID}_v, 0, \perp) \tag{3.11}$$

Le prédicat RootBFS$(v)$ permet de vérifier que le nœud $v$ est la racine de l'arbre BFS. En d'autres termes, il vérifie si l'identifiant de la racine du BFS est égale à son identifiant, si il n'a pas de parent et si sa distance est égale à zéro.

$$\text{RootBFS}(v) \equiv (\mathsf{R}_{\text{BFS}v}, \mathsf{P}_{\text{BFS}v}, \mathsf{d}_{\text{BFS}v}) = (\text{ID}(v), \perp, 0) \tag{3.12}$$

Sinon, la règle $\mathbb{R}_{\texttt{StartBFS}}$ assigne les variables $\mathsf{R}_{\text{BFS}v}, \mathsf{P}_{\text{BFS}v}, \mathsf{d}_{\text{BFS}v}$ dans ce sens.

La règle $\mathbb{R}_{\texttt{Tree}}$ est consacrée à propager l'identifiant de la racine $r$ de l'arbre couvrant BFS courant le long du `Backbone` par les nœuds qui ne détiennent pas le jeton. Cette inondation est possible puisque nous réorientons le `Backbone` en $r$. La règle $\mathbb{R}_{\texttt{Tree}}$ détecte également les erreurs locales.

$$\mathbb{R}_{\texttt{Tree}} : \neg\text{GoodBFS}(v) \longrightarrow (\mathsf{R}_{\text{BFS}v}, \mathsf{P}_{\text{BFS}v}, \mathsf{d}_{\text{BFS}v}, \mathsf{D}_{\text{BFS}v}) := (\mathsf{R}_{\text{BFS}_{\text{p\_next}(v)}}, \perp, \infty, \perp) \tag{3.13}$$

Le prédicat GoodBFS($v$) vérifie si la racine BFS est la même que celle de son parent dans le `Backbone` qui mène au jeton. En d'autres termes, $\mathsf{R}_{\text{BFS}v}$ doit être égale à l'identifiant du nœud avec le jeton.

$$\text{GoodBFS}(v) \equiv (\mathsf{R}_{\text{BFS}v} = \mathsf{R}_{\text{BFS}_{\text{p\_next}_v}}) \tag{3.14}$$

Pour calculer le BFS actuel et l'excentricité, tout les nœuds doivent avoir comme racine BFS l'identifiant du nœud qui garde le jeton. Ainsi, toutes les autres règles vérifient GoodBFS($v$). La règle $\mathbb{R}_{\texttt{BFS}}$ exécute la construction BFS en elle-même.

$$\begin{aligned} \mathbb{R}_{\texttt{BFS}} \quad : \quad & \text{GoodBFS}(v) \wedge \text{SameBFS}(v) \wedge (\text{Best}(v) \neq \perp) \\ & \longrightarrow \text{ if } \mathsf{D}_{\text{BFS}v} \in \mathbb{N} \text{ alors } \mathsf{D}_{\text{BFS}v} :=\uparrow;\ \text{sinon } \mathsf{D}_{\text{BFS}v} :=\downarrow; \\ & \longrightarrow (\mathsf{P}_{\text{BFS}v}, \mathsf{d}_{\text{BFS}v}) := (\text{Best}(v), \mathsf{d}_{\text{BFS}_{\text{Best}(v)}} + 1); \end{aligned} \tag{3.15}$$

Le prédicat SameBFS($v$) vérifie si tout les voisins de $v$ sont dans le même arbre BFS que $v$. En d'autres termes, si tous ses voisins ont la même racine que celui-ci.

$$\text{SameBFS}(v) \equiv (\forall u \in N_v, (\mathsf{R}_{\text{BFS}u} = \mathsf{R}_{\text{BFS}v})) \tag{3.16}$$

La dernière fonction de la règle $\mathbb{R}_{\texttt{BFS}}$, le prédicat Best($v$) vérifie si le nœud $v$ a un voisin avec une meilleure distance, sinon elle retourne $\perp$.

$$\text{Best}(v) = \begin{cases} p = \min\{\text{ID}(u) \mid \mathsf{d}_{\text{BFS}u} = \min\{\mathsf{d}_{\text{BFS}w} \mid w \in N_v\}\} \text{ si } \big((p \neq \mathsf{P}_{\text{BFS}v}) \vee (\mathsf{d}_{\text{BFS}v} \neq \mathsf{d}_{\text{BFS}p} + 1)\big) \\ \perp \text{ sinon} \end{cases} \tag{3.17}$$

Le nœud $v$ peut nettoyer sa variable uniquement si sa distance est une valeur entière.

Si le nœud $v$ a un meilleur parent $p$ dans le courant BFS, $v$ modifie ses variables relatives au BFS en accord avec les variables de $p$.

La règle $\mathbb{R}_{\texttt{Ecc}}$ traite de la phase de collecte de l'excentricité des feuilles à la racine.

$$\begin{aligned} \mathbb{R}_{\texttt{Ecc}} \quad : \quad & \text{GoodBFS}(v) \wedge \text{SameBFS}(v) \wedge (\forall u \in N_v : \mathsf{d}_{\text{BFS}u} \neq \infty) \wedge (\text{Best}(v) = \perp) \wedge (\mathsf{D}_{\text{BFS}v} \neq \text{Dis}(v)) \\ & \longrightarrow \mathsf{D}_{\text{BFS}v} := \text{Dis}(v) \end{aligned} \tag{3.18}$$

En raison de l'asynchronisme du système, le nœud $v$ peut stocker une excentricité erronée. Pour gérer cela, nous ajoutons trois valeurs à $\mathsf{D}_{\text{BFS}v}$, les valeurs $\{\perp, \uparrow, \downarrow\}$. Plus précisément :

- La variable $\mathsf{D}_{\text{BFS}v}$ a la valeur $\perp$ quand l'identifiant du nouveau BFS est propagé dans `Backbone` ou quand un nettoyage est effectué de la racine du `BFS` vers les feuilles.
- La variable $\mathsf{D}_{\text{BFS}v}$ a la valeur $\downarrow$ lorsque le nœud peut calculer la distance maximale du sous-arbre vers la racine du `BFS` si et seulement si ses enfants ont eux mêmes calculé leurs distances maximales.
- La variable $\mathsf{D}_{\text{BFS}v}$ a la valeur $\uparrow$ lorsque le nœud a détecté une incoherence dans la distance maximale recueillie ou lorsque un de ses voisins a lui même détecté une incohérence.

L'affectation de $\mathsf{D}_{\text{BFS}v}$ est gérée par la commande $\text{Dis}(v)$. Nous allons décrire plus en détails le mécanisme pour éviter le calcul d'une excentricité erronée due à l'asynchronisme. L'excentricité du nœud $v$ est la plus grande distance de ses descendants, cette distance est calculée en partant des feuilles vers la racine. Lorsqu'un nœud $v$ est une feuille ou que tout les enfants d'un nœud $v$ dans le BFS courant ont calculé la distance la plus éloignée de leurs descendants, le nœud $v$ peut calculer sa propre distance $\mathsf{D}_{\text{BFS}v}$. Il est à noter que si l'un des voisins de $v$ n'a pas calculé de distance, il ne peut effectuer ce calcul. Le prédicat $\text{Ch\_bfs}(v)$ renvoie les enfants de $v$ dans le BFS courant. En d'autres termes, les voisins $u$ avec la même racine BFS, la variable parent de $u$ indique $v$ et la distance de $u$ est la distance de $v$ plus un.

$$\text{Ch\_bfs}(v) = \{u \mid u \in N_v \wedge (\mathsf{R}_{\text{BFS}u}, \mathsf{P}_{\text{BFS}u}, \mathsf{d}_{\text{BFS}u}) = (\mathsf{R}_{\text{BFS}v}, v, \mathsf{d}_{\text{BFS}v} + 1)\} \tag{3.19}$$

La plus grande distance du nœud $v$ est calculée grâce à la fonction $\mathsf{MaxD}(v)$. Pour ce faire, $v$ compare sa distance et les distances plus lointaines retournées par ses enfants dans le BFS courant.

$$\mathsf{MaxD}(v) = \max\{\mathsf{d}_{\mathsf{BFS}_v}, \max\{\mathsf{D}_{\mathsf{BFS}_u} \mid u \in \text{Ch\_bfs}(v)\}\} \tag{3.20}$$

En raison de l'asynchronisme du système, un nœud peut calculer sa plus grande distance alors que la construction du BFS courant n'est pas terminée (voir Figure 3.3(a)). Quand un nœud $v$ change de distance (et de parent), il met sa variable $\mathsf{D}_{\mathsf{BFS}_v}$ à $\downarrow$ (voir la règle $\mathbb{R}_{\mathtt{BFS}}$( 3.15) et Figure 3.3(b)). Donc, si l'ancien parent de $v$ a déjà calculé sa distance la plus éloignée, il devient incohérent (voir Figure 3.3(c)). Le prédicat $\mathsf{CohD}_E(v)$ est consacré à vérifier ce type de cohérence. Pour être cohérent, le nœud $v$ vérifie d'abord si tous ses enfants dans le `BFS` courant ont déjà calculé la distance la plus éloignée. Si oui, la variable $\mathsf{D}_{\mathsf{BFS}_v}$ doit être égale à la distance la plus éloignée calculée par la fonction $\mathsf{MaxD}(v)$. Dans le cas de la détection d'une incohérence (voir Figure 3.3(c)), les voisins de ce nœud $v$ (avec $\mathsf{D}_{\mathsf{BFS}_v} = \uparrow$) vont répercuter cette incohérence pour nettoyer l'arbre des distances maximales erronées. Autrement dit, tout les nœuds $u$ du `BFS` vont passer à $\mathsf{D}_{\mathsf{BFS}_u} = \uparrow$ à l'exception de la racine qui passe à $\perp$ (voir Figure 3.3(d)). La valeur $\perp$ se propage maintenant de la racine vers les feuilles du `BFS` (voir Figure 3.3(e)). Quand un nœud et ses enfants dans le `BFS` ont été nettoyés, il peut repasser en $\downarrow$ (voir Figure 3.3(f)). Lorsque les feuilles du `BFS` ont rejoint $\downarrow$, le calcul des distances maximales peut reprendre (voir Figure 3.3(g)).

$$\begin{aligned}\mathsf{CohD}_E(v) \equiv (\mathsf{D}_{\mathsf{BFS}_v} \in \mathbb{N}) \wedge (\mathsf{d}_{\mathsf{BFS}_v} = \min\{\mathsf{d}_{\mathsf{BFS}_u} : \forall u \in N_v\} + 1)\\ \wedge(\forall u \in \text{Ch\_bfs}(v), \mathsf{D}_{\mathsf{BFS}_u} \in \mathbb{N}) \wedge (\mathsf{D}_{\mathsf{BFS}_v} = \mathsf{MaxD}(v)) \wedge (\mathsf{D}_{\mathsf{BFS}_{\mathsf{P}_{\mathsf{BFS}_v}}} \in \{\downarrow, \mathbb{N}\})\end{aligned} \tag{3.21}$$

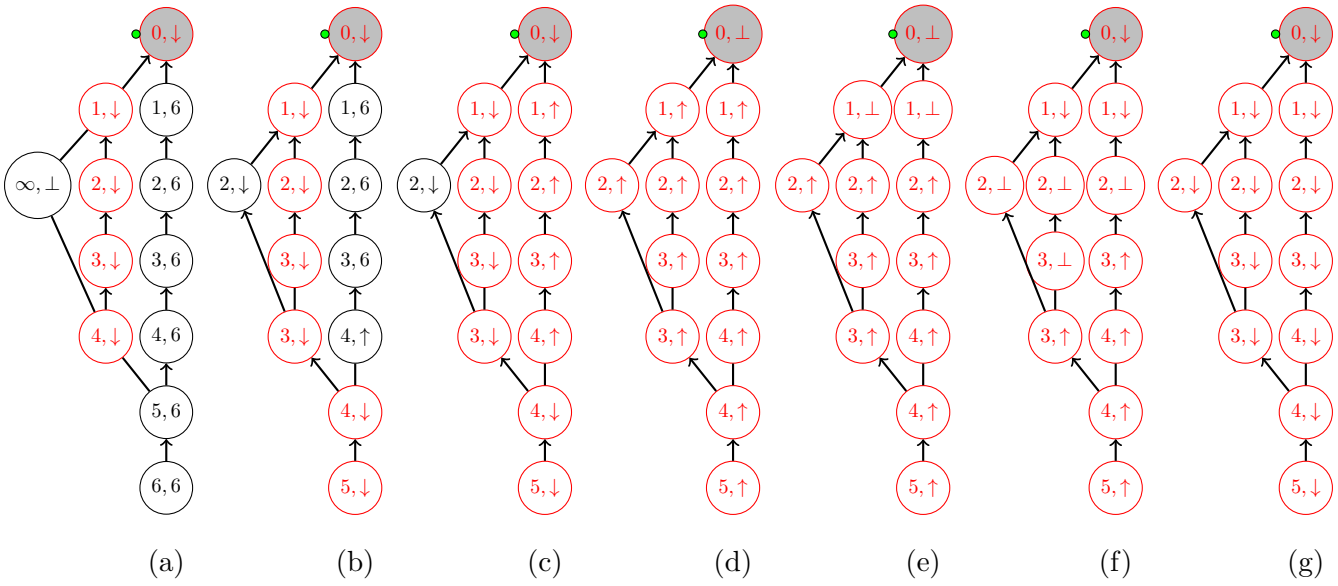



FIGURE 3.3 – Example de calcul de l'excentricité.
La première valeur dans le nœud est la distance à partir de la racine, la dernière valeur est la variable $\mathsf{D}_{\mathsf{BFS}_v}$. En rouge les chemins bloquants, notion définie dans les preuves.

En outre, pour être cohérent, un nœud $v$ doit vérifier la cohérence de sa variable $\mathsf{D}_{\mathsf{BFS}_v}$ avec celle de son parent dans le BFS courant $\mathsf{D}_{\mathsf{BFS}_{\mathsf{P}_{\mathsf{BFS}_v}}}$, c'est la règle du prédicat $\mathsf{CohD}_{\bar{N}}(v)$.

$$\mathsf{CohD}_{\bar{N}}(v) \equiv (\mathsf{D}_{\mathsf{BFS}_v}, \mathsf{D}_{\mathsf{BFS}_{\mathsf{P}_{\mathsf{BFS}_v}}}) \in \{(\perp,\perp),(\uparrow,\perp),(\perp,\downarrow),(\uparrow,\uparrow),(\downarrow,\downarrow)\} \tag{3.22}$$

Par conséquent, un nœud est cohérent si il a $\mathsf{CohD}_{\bar{N}}(v)$ et $\mathsf{CohD}_E(v)$.

$$\mathsf{CohD}(v) \equiv \mathsf{CohD}_{\bar{N}}(v) \vee \mathsf{CohD}_E(v) \tag{3.23}$$

Donc, si un nœud $v$ est incohérent, $\mathsf{D}_{\mathsf{BFS}_v}$ devient $\uparrow$ et tous ses voisins prennent les mêmes valeurs (voir Figure 3.3(c)), jusqu'aux voisins de la racine du BFS actuel. La deuxième ligne de la commande $\mathrm{Dis}(v)$ assure ce mécanisme.

$$\mathrm{Dis}(v) = \begin{cases} \perp & \text{if } (\mathsf{D}_{\mathsf{BFS}_v} = \uparrow) \wedge (\mathsf{D}_{\mathsf{BFS}_{\mathsf{P}_{\mathsf{BFS}_v}}} = \perp) \wedge (\forall u \in \mathrm{Ch\_bfs}(v), \mathsf{D}_{\mathsf{BFS}_u} = \uparrow) \\ \uparrow & \text{if } \neg\mathsf{CohD}(v) \vee \big(\mathsf{CohD}(v) \wedge (\exists u \in N_v \mid \mathsf{D}_{\mathsf{BFS}_u} = \uparrow)\big) \\ \downarrow & \text{if } \mathsf{CohD}(v) \wedge (\mathsf{D}_{\mathsf{BFS}_v} = \perp) \wedge (\forall u \in \mathrm{Ch\_bfs}(v), \mathsf{D}_{\mathsf{BFS}_u} = \perp) \wedge (\mathsf{D}_{\mathsf{BFS}_{\mathsf{P}_{\mathsf{BFS}_v}}} = \downarrow) \\ \mathsf{MaxD}(v) & \text{if } \mathsf{CohD}(v) \wedge (\mathsf{D}_{\mathsf{BFS}_v} = \downarrow) \wedge (\forall u \in \mathrm{Ch\_bfs}(v), \mathsf{D}_{\mathsf{BFS}_u} \in \mathbb{N}) \end{cases} \tag{3.24}$$

Une fois que tout les voisins $v$ de la racine $r$ du `BFS` atteignent $\mathsf{D}_{\mathsf{BFS}_v} = \uparrow$, la règle $\mathbb{R}_{\texttt{CleanEcc}}(r)$ prend le dessus et la racine $r$ qui avait $\mathsf{D}_{\mathsf{BFS}_r} = \downarrow$ met sa variable $\mathsf{D}_{\mathsf{BFS}}$ à $\perp$ pour nettoyer la variable $\mathsf{D}_{\mathsf{BFS}}$ des nœuds du `BFS` (voir prédicat $\mathrm{Down}(v)$ et la règle $\mathbb{R}_{\texttt{CleanEcc}}(r)$).

$$\mathrm{Down}(v) \equiv (\mathsf{D}_{\mathsf{BFS}_v} = \downarrow) \wedge (\forall u \in N_v, \mathsf{D}_{\mathsf{BFS}_u} = \uparrow) \tag{3.25}$$

$$\begin{aligned} \mathbb{R}_{\texttt{CleanEcc}}(v) \quad : \quad & \mathrm{RootBFS}(v) \wedge \mathrm{SameBFS}(v) \wedge (\mathrm{Down}(v) \vee \mathrm{Bot}(v)) \\ & \longrightarrow \mathsf{D}_{\mathsf{BFS}_v} = \perp \text{ if } \mathrm{Down}(v) \\ & \longrightarrow \mathsf{D}_{\mathsf{BFS}_v} = \downarrow \text{ if } \mathrm{Bot}(v) \end{aligned} \tag{3.26}$$

De la racine du BFS aux feuilles, tout les nœuds $v$ prennent les valeurs $\mathsf{D}_{\mathsf{BFS}_v} = \perp$ grâce à la première ligne de la commande $\mathrm{Dis}(v)$. Un nœud $v$ peut prendre la valeur $\mathsf{D}_{\mathsf{BFS}_v} = \perp$ uniquement si tout ses enfants ont $\mathsf{D}_{\mathsf{BFS}_u} = \uparrow$. Ceci nous garantit que le nœud $v$ soit à $\mathsf{D}_{\mathsf{BFS}_v} = \perp$ alors que ses enfants ont calculé leurs excentricité. Quand les enfants de la racine du `BFS` sont à $\perp$, la racine peut se mettre à $\downarrow$ (prédicat $\mathrm{Bot}(v)$), ce qui permettra à tout les nœuds du `BFS` de passer à $\downarrow$ et de permettre par conséquent le calcul de l'excentricité de la racine du `BFS`.

$$\mathrm{Bot}(v) \equiv (\mathsf{D}_{\mathrm{BFS}v} = \bot) \wedge (\forall u \in N_v, \mathsf{D}_{\mathrm{BFS}u} = \bot) \tag{3.27}$$

Enfin, lorsque tout les nœuds $v$ ont $\mathsf{D}_{\mathrm{BFS}v} =\downarrow$, la règle $\mathbb{R}_{\mathtt{Ecc}}$ collecte la plus grande distance à partir des feuilles vers la racine (voir aussi la dernière ligne de $\mathrm{Dis}(v)$).

La règle $\mathbb{R}_{\mathtt{EndBFS}}$ est exécutée par la racine du BFS lorsque la propagation de l'excentricité des feuilles vers la racine est terminée. Ceci est vérifié par le prédicat $\mathrm{ChN}(v)$ dans la règle $\mathbb{R}_{\mathtt{EndBFS}}$. En d'autres termes, tout les enfants de la racine ont déjà calculé leur distance la plus éloignée.

$$\mathrm{ChN}(v) \equiv \{\forall u \in N_v, \mathsf{D}_{\mathrm{BFS}u} \in \mathbb{N}\} \tag{3.28}$$

Cette règle calcule l'excentricité de $r$, libère le jeton en mettant à jour la variable $\mathsf{next}_r$ de la couche de circulation du jeton (voir la section 3.4.2) et met à jour la variable $\mathsf{MinEcc}$ pour communiquer la nouvelle excentricité à la couche de calcul des centres (voir la section 3.4.4 pour plus de détails sur l'utilisation de la variable $\mathsf{MinEcc}(v)$).

$$\begin{aligned} \mathbb{R}_{\mathtt{EndBFS}} : \mathrm{RootBFS}(v) \wedge \mathrm{SameBFS}(v) \wedge \mathrm{ChN}(v) \\ \longrightarrow \mathsf{Ecc}_v := \mathrm{Dis}(v); \mathsf{next}_v := \mathrm{FNext}(v); \\ \mathsf{MinEUP}_v := \mathsf{MinEcc}(v); \end{aligned} \tag{3.29}$$

### 3.4.4 Calcul des centres

La quatrième et dernière couche de notre algorithme vise à identifier les centres du système. Étant donné que chaque nœud calcule sa propre excentricité avec les trois premières couches de notre algorithme, il ne reste plus qu'à calculer l'excentricité minimal. Pour faire cela, nous utilisons le `Backbone` (orienté vers la racine désignée par la première couche). Tout d'abord, la racine collecte l'excentricité minimale dans le système par une vague partant des feuilles et arrivant à la racine. Ensuite, la racine propage cette excentricité minimum sur le `Backbone`. Cet algorithme fonctionne sous le démon distribué non équitable, utilise $O(\log n)$ bits de mémoire par nœud, et se stabilise en $O(n)$ rondes.

Cette couche utilise les variables suivantes :

— La variable $\mathsf{Ecc}_v$ (maintenue par la troisième couche, voir la section 3.4.3) stocke l'excentricité du nœud $v$.

— La variable $\mathsf{MinEUP}_v$ est utilisée pour collecter l'excentricité minimale des feuilles vers la racine.

— La variable $\mathsf{MinE}_v$ est utilisée pour stocker l'excentricité minimale du système et elle sert à sa diffusion de la racine vers les feuilles.

Nous définissons la fonction suivante pour calculer le minium des excentricités d'un nœud $v$ par rapport à sa propre variable $\mathsf{Ecc}_v$ et celles de ses enfants :

**Algorithm 4:** Calcul de l'excentricité minimum pour un nœud $v$

$$\mathbb{R}_{\texttt{MinEUp}} \quad : \quad \mathsf{MinEUP}_v \neq \mathsf{MinEcc}(v) \longrightarrow \mathsf{MinEUP}_v := \mathsf{MinEcc}(v);$$

$$\mathbb{R}_{\texttt{MinERoot}} \quad : \quad \mathrm{BRoot}(v) \wedge (\mathsf{MinE}_v \neq \mathsf{MinEUP}_v) \longrightarrow \mathsf{MinE}_v := \mathsf{MinEUP}_v;$$

$$\mathbb{R}_{\texttt{MinEDown}} \quad : \quad \neg\mathrm{BRoot}(v) \wedge (\mathsf{MinE}_v \neq \mathsf{MinE}_{\mathsf{p}_v}) \longrightarrow \mathsf{MinE}_v := \mathsf{MinE}_{\mathsf{p}_v};$$

$$\mathsf{MinEcc}(v) = \min\{\mathsf{Ecc}_v, \min\{\mathsf{MinE}_u \mid u \in \mathrm{child}(v)\} \tag{3.30}$$

Une présentation formelle de cette couche est donnée dans l'Algorithme 4. La règle $\mathbb{R}_{\texttt{MinEUp}}$ collecte des feuilles vers la racine l'excentricité minimal alors que la règle $\mathbb{R}_{\texttt{MinERoot}}$ et la règle $\mathbb{R}_{\texttt{MinEDown}}$ assurent la propagation de la racine vers les feuilles de l'excentricité minimale calculée par la racine.

Une fois que cet algorithme se stabilise, chaque nœud connaît son excentricité (grâce à la troisième couche) et l'excentricité minimale dans le système (grâce à la quatrième couche). Ensuite, il est trivial pour un nœud de décider si il est un centre ou non. Comme nous supposons que le système est non anonyme, il est facile de choisir le centre avec l'identifiant minimale dans le cas où le système admet plus d'un centre pour construire un unique arbre couvrant de diamètre minimum. Cette phase nécessite $O(\log n)$ bits de mémoire par nœud et stabilise au bout de $O(n)$ rondes.

En conclusion, la composition de ces quatre couches nous fournit un algorithme auto-stabilisant pour le calcul des centres ou pour la construction d'un arbre couvrant de diamètre minimum sous un démon distribué non équitable qui nécessite $O(\log n)$ bits de mémoire par nœud et se stabilise en $O(n^2)$ rondes.

### 3.4.5 Ordre de priorité entre les règles de notre algorithme

La Figure 3.4 illustre d'ordre des priorités entre nos différentes couches. Elle donne plus précisément l'ordre de priorité entre les règles, celles liés à la construction du `Backbone` sont les règles les plus prioritaires. Nous précisons que les règles du calcul du centre sont prioritaires par rapport au calcul de l'excentricité de chaque nœud. Ces priorités entre les règles nous assurent la convergence de notre algorithme.

## 3.5 Preuve de l'algorithme

Une façon classique de prouver l'auto-stabilisation d'un algorithme est de prouver que l'ensemble des configurations légitimes (les configurations répondant à la spécification du problème) est un *attracteur* de $\Gamma$ pour cet algorithme. Étant donné deux ensembles de configurations $\Gamma_2 \subseteq \Gamma_1 \subseteq \Gamma$, on dit que $\Gamma_2$ est un attracteur de $\Gamma_1$ pour l'algorithme $\mathcal{A}$ (noté $\Gamma_1 \rhd \Gamma_2$) si une exécution de $\mathcal{A}$ à partir de toute configuration de $\Gamma_1$ atteint en un temps fini une configuration de $\Gamma_2$ et si $\Gamma_2$ est clos sous $\mathcal{A}$. Il est important de préciser que comme notre algorithme est non-silencieux, la clôture de certains attracteurs ne sera pas démontrée.

Ces preuves sont réalisées en utilisant des *fonctions de potentiels*. Une fonction de

potentiel est une fonction bornée qui associe à chaque configuration du système une valeur qui diminue strictement à chaque application d'une règle par l'algorithme. Par conséquent, si la valeur minimale de la fonction est associée à une configuration légitime du système, l'existence d'une telle fonction est suffisante pour prouver la convergence de l'algorithme. La difficulté est évidemment de mettre en évidence la fonction de potentiel qui capture avec précision le comportement de l'algorithme auto-stabilisant. [HC92] ont proposé une fonction de potentiel pour la construction d'un BFS mais celle-ci ne convient pas dans notre cas (en raison d'une caractérisation trop faible des effets de l'algorithme sur les configurations et l'utilisation d'une *a priori* connaissance des nœuds). C'est la raison pour laquelle nous proposons une fonction de potentiel plus adaptée pour notre algorithme pour prouver le lemme 11.

**Théorème 1** *L'algorithme $\mathcal{SSCC}$ est un algorithme auto-stabilisant qui calcule le(s) centre(s) du système sous un démon distribué non équitable. Il utilise $O(\log n)$ bits de mémoire par nœud et se stabilise en $O(n^2)$ rondes.*

Notons $\Gamma_{\texttt{Backbone}} \subseteq \Gamma$ comme l'ensemble des configurations de $\Gamma$ de telle sorte qu'aucune règle de la première couche de $\mathcal{SSCC}$ (élection et construction du `Backbone`) ne soit activable.

**Lemme 1** $\Gamma \triangleright \Gamma_{\texttt{Backbone}}$ *en $O(n^2)$ rondes pour $\mathcal{SSCC}$ sous un démon distribué non équitable.* $\Gamma_{\texttt{Backbone}}$ *est clos.*

**Preuve.**

La preuve de ce lemme est en partie basée sur celle de l'algorithme de construction du `Backbone` choisit. Pour rappel, nous ne modifions pas cet algorithme et nous assumons que cette couche de notre algorithme a la plus haute priorité. Pour la clôture de $\Gamma_{\texttt{Backbone}}$ il faut donc se rapporter aux preuves de l'algorithme choisit. Il nous reste donc à prouver la convergence dans $\Gamma_{\texttt{Backbone}}$ de notre algorithme. Autrement dit, que les règles de

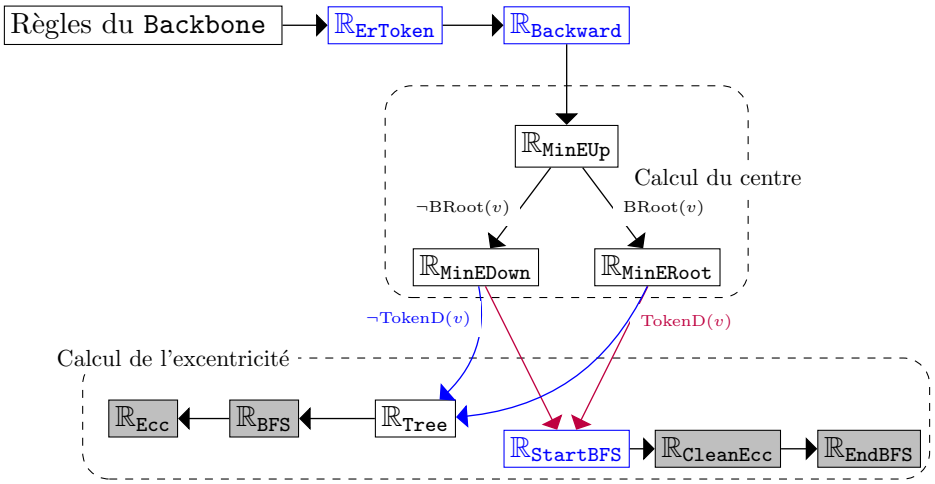



FIGURE 3.4 – Ordre de priorité des règles.
En bleu les règles de la circulation du jeton. Les règles sur font gris exigent toutes que le prédicat SameBFS soit vrai.

notre algorithme $\mathcal{SSCC}$ non dédiées à la construction du `Backbone` n'empêchent pas sa construction. Pour cela nous donnons une preuve par contradiction, nous supposons que le démon n'active jamais les règles liées à la construction du `Backbone`. Si le démon est obligé d'activer les règles de construction du `Backbone`, alors nous convergeons vers $\Gamma_{\texttt{Backbone}}$.

Considérons une configuration initale $\gamma_0 \notin \Gamma_{\texttt{Backbone}}$ et soit $A(\gamma_0)$ l'ensemble des nœuds activables par une règle. Notons par $B(\gamma_0)$ les nœuds activables par les règles de construction du `Backbone` dans $\gamma_0$. Le démon voulant contrer la construction du `Backbone`, il va donc essayer de ne jamais activer les nœuds éléments de $B(\gamma_0)$. Remarquons $|B(\gamma_0)| < n$ sinon le `Backbone` serait construit. De plus $B(\gamma_0) \subset A(\gamma_0)$ dû au l'ordre de priorité des règles de l'algorithme, la Figure 3.4 donne l'ordre des priorités entre les règles de notre algorithme. Enfin $A(\gamma_0) \setminus B(\gamma_0) \neq \emptyset$, sinon le démon devra activer les nœuds activables par les règles de construction du `Backbone`. Nous allons donc prouver que quelque soit la stratégie d'activation des nœuds du démon, le système va converger vers une configuration $\gamma$ tel que $A(\gamma) \setminus B(\gamma) = \emptyset$ avec $B(\gamma)$ non vide.

Dans une configuration initiale $\gamma_0$ il peut exister trois types de sous-structures couvrantes induites par la variable parent de chaque nœud destiné à construire le `Backbone`.

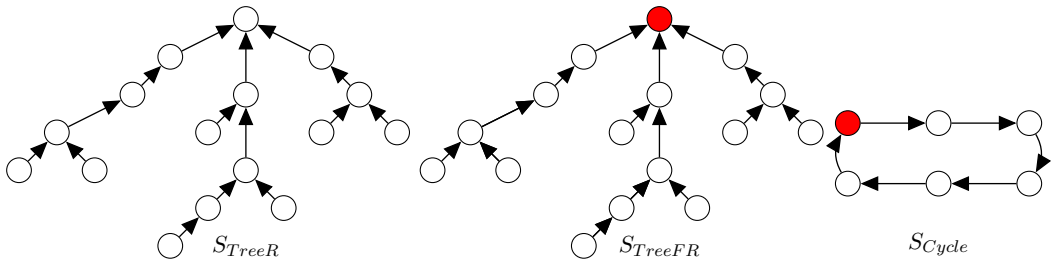



FIGURE 3.5 – Sous-structures couvrantes possibles en l'absence de construction du `Backbone`. Les nœuds rouges sont activables par au moins une règle de construction du `Backbone`.

$S_{TreeR}$ : La sous-structure couvrante $S_{TreeR}$ est un sous-arbre enraciné en un nœud $v$ qui ne détecte pas d'erreur de construction du `Backbone`, soit parce que $v$ a l'identifiant le plus petit du réseau, soit que l'identifiant de $v$ est le plus petit identifiant dans son voisinage. Nous précisons que $v$ n'est pas activable par une règle de construction du `Backbone`. Sans perte de généralité, nous supposons $\forall u \in S_{TreeR}, u \notin B(\gamma_0)$.

$S_{TreeFR}$ : La sous-structure couvrante $S_{TreeFR}$ est un sous-arbre enraciné en un nœud qui détecte une erreur dans la construction du `Backbone`, $v$ est activable par une règle de construction du `Backbone`. Sans perte de généralité, nous supposons que seul $v$ la racine $S_{TreeFR}$ est élément de $B(\gamma_0)$, les autres nœuds de $S_{TreeFR}$ ne sont pas éléments de $B(\gamma_0)$.

$S_{Cycle}$ : La sous-structure couvrante $S_{Cycle}$ contient un cycle. Il existe donc au moins un nœud $v$ qui détecte la présence d'un cycle, $v$ est activable par une règle de construction du `Backbone`. Sans perte de généralité, nous supposons que seul $v$ est élément de $B(\gamma_0)$, les autres nœuds de $S_{Cycle}$ ne sont pas éléments de $B(\gamma_0)$.

Nous allons donc nous intéresser à la règle $\mathbb{R}_{\texttt{ErToken}}$ en premier puisque elle est directement prioritaire après les règles relatives à la construction du `Backbone`. Il est à noter que si le démon décide de ne pas activer les nœuds disponibles pour la règle $\mathbb{R}_{\texttt{ErToken}}$, ces nœuds ne seront activables par aucune autre règle à cause de l'ordre des priorités des règles

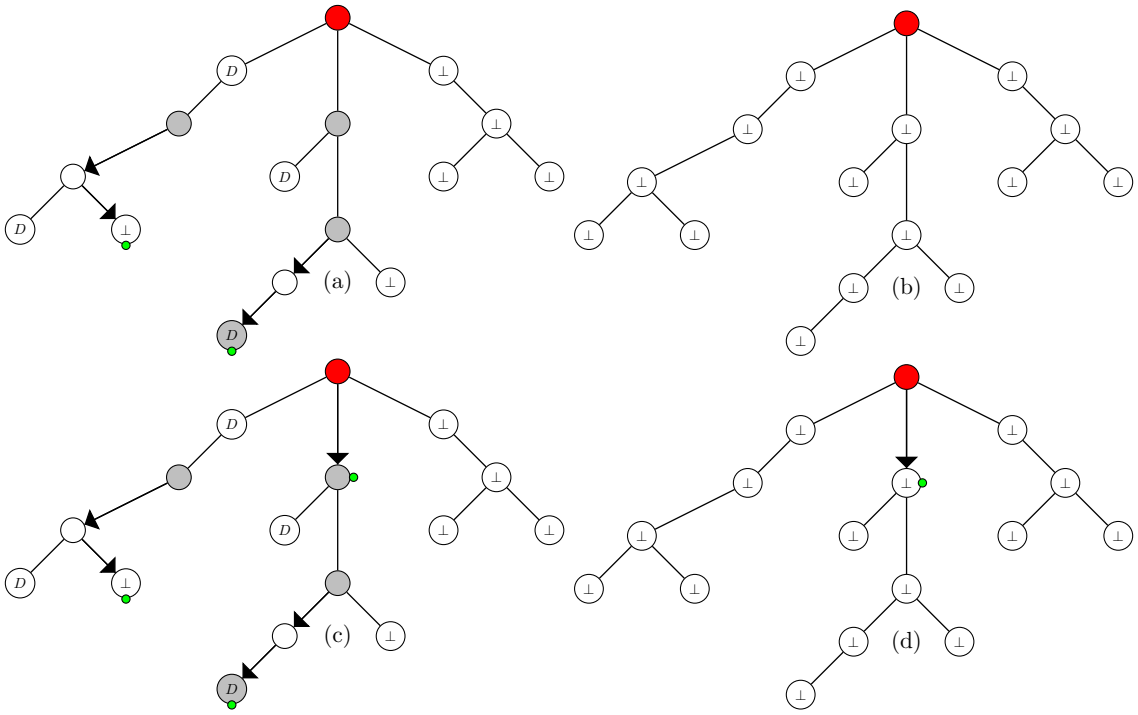



FIGURE 3.6 – Application de la règle $\mathbb{R}_{\texttt{ErToken}}$ dans un sous-arbre $S_{TreeFR}$. Les nœuds gris sont activables par la règle $\mathbb{R}_{\texttt{ErToken}}$.

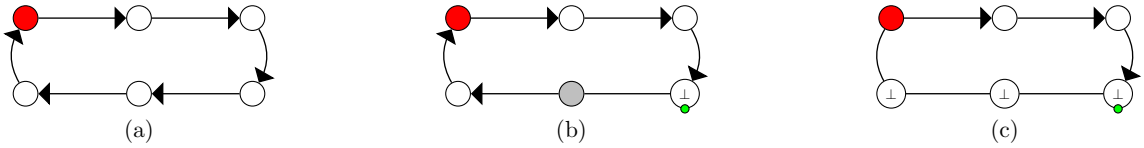



FIGURE 3.7 – Application de la règle $\mathbb{R}_{\texttt{ErToken}}$ dans une structure couvrante $S_{Cycle}$. Les nœuds gris sont activables par la règle $\mathbb{R}_{\texttt{ErToken}}$.

(voir Figure 3.4). De plus, aucune action de leurs voisins (autre que la règle $\mathbb{R}_{\texttt{ErToken}}$) les rendront non activables pour la règle $\mathbb{R}_{\texttt{ErToken}}$. Il en va de même pour la règle $\mathbb{R}_{\texttt{Backward}}$.

**Propriété** 1 ( $\mathbb{R}_{\texttt{ErToken}}$) *A partir d'une configuration* $\Gamma \not\subset \Gamma_{\texttt{Backbone}}$*, le nombre d'activation de la règle* $\mathbb{R}_{\texttt{ErToken}}$ *est borné.*

**Preuve de la propriété.** Nous ne changeons pas la règle $\mathbb{R}_{\texttt{ErToken}}$ (3.1) par rapport à l'algorithme de [PV99]. La preuve du théorème 3 de [PV99] dit que toute exécution à partir d'une configuration qui est un arbre atteint une configuration avec un unique jeton en $O(|T|)$ rondes (où $T$ est l'arbre où le sous-arbre considéré). Autrement dit, une configuration où $\mathbb{R}_{\texttt{ErToken}}$ n'est plus activable.

$S_{TreeR}$ : $S_{TreeR}$ est un sous-arbre, la preuve du théorème 3 de [PV99] s'applique et on obtient un seul jeton par $S_{TreeR}$.

$S_{TreeFR}$ : La sous-structure couvrante $S_{TreeFR}$ est un sous-arbre enraciné en un nœud $v$, la preuve du théorème 3 de [PV99] s'applique. Le nœud $v$ est élément de $B(\gamma_0)$, par

hypothèse le démon ne l'active pas. L'algorithme de [PV99] maintient un chemin de la racine vers le nœud possédant le jeton. On obtient donc deux cas de figure.

Soit la racine $v$ avait sa variable $\mathsf{next}_v \in \{\perp, \text{done}\}$ (voir Figure 3.6(a)), si le démon active jusqu'à ce que cela ne soit plus possible la règle $\mathbb{R}_{\texttt{ErToken}}$ dans $S_{TreeFR}$ on obtient une structure $S_{TreeFR}$ sans jeton (voir Figure 3.6(b)). En l'absence de jeton, seule la racine de l'arbre crée un nouveau jeton. Dans notre cas le jeton ne peut pas être créé par la racine car la racine est dans l'ensemble $B(\gamma_0)$ et que nous avons supposé que le démon n'activé pas les nœuds de cet ensemble.

Soit la variable $\mathsf{next}_v$ pointe vers un de ses enfant (voir Figure 3.6(c)), si le démon active jusqu'à cela ne soit plus possible la règle $\mathbb{R}_{\texttt{ErToken}}$ dans $S_{TreeFR}$ on obtient une structure $S_{TreeFR}$ avec un unique jeton (voir Figure 3.6(d)).

$S_{Cycle}$ : La sous-structure couvrante $S_{Cycle}$ contient un cycle. Si tout les nœuds $v$ du cycle ont leurs variables $\mathsf{next}_v$ qui pointent vers leur enfant dans le cycle, alors aucun nœud ne sera activable par la règle $\mathbb{R}_{\texttt{ErToken}}$, de plus le cycle ne contient pas de jeton (voir Figure 3.7(a)). Il n'y aura pas création de jeton car il n'existe pas de racine dans le cycle, et qu'en l'absence de jeton seule la racine peut en créer un.

Si il existe un nœud $v$ dans le cycle dont la variable $\mathsf{next}_v$ ne pointe pas vers son enfant (voir Figure 3.7(b)), alors tout les nœuds de $S_{Cycle}$ entre le nœud $v$ et le nœud $u$ (le nœud activable par une règle de la construction du `Backbone`) exécuterons la règle $\mathbb{R}_{\texttt{ErToken}}$. Le nœud $w$ prédécesseur de $v$ considère qu'il possède l'unique jeton du cycle (voir Figure 3.7(b)) .

En conclusion, les nœuds $v \in V$ ne peuvent appliquer qu'une et une seule fois la règle $\mathbb{R}_{\texttt{ErToken}}$. ■

Nous avons vu que le nombre d'activation de la règle $\mathbb{R}_{\texttt{ErToken}}$ était borné, nous considérons donc maintenant une configuration $\gamma$. Soit $X(\gamma)$ les nœuds activables par la règle $\mathbb{R}_{\texttt{ErToken}}$, cet ensemble peut-être nul, soit parce que le démon les a tous activés, soit parce qu'aucun nœud n'était activable par la règle $\mathbb{R}_{\texttt{ErToken}}$ dans la configuration initiale. Nous considérons donc un ensemble $A(\gamma)$ composé des nœuds activables, avec $B(\gamma) \subset A(\gamma)$, $X(\gamma) \subset A(\gamma)$ et $A(\gamma) \setminus B(\gamma) \cup X(\gamma) \neq \emptyset$.

**Propriété** 2 ($\mathbb{R}_{\texttt{Backward}}$) *À partir d'une configuration $\gamma \not\subset \Gamma_{\texttt{Backbone}}$, le nombre d'activation consécutive de la règle $\mathbb{R}_{\texttt{Backward}}$ est borné.*

**Preuve de la propriété.** Soit $v$ un nœud activable par la règle $\mathbb{R}_{\texttt{Backward}}$ ( 3.3), si $v$ possède un ascendant $u \in A(\gamma)$ dans la sous-structure couvrante (sous-arbre ou cycle), le nombre d'activation de la règle $\mathbb{R}_{\texttt{Backward}}$ est limité par la hauteur entre $v$ et le premier ascendant de $v$ qui est élément de $B(\gamma) \cup X(\gamma)$. La règle $\mathbb{R}_{\texttt{ErToken}}$ et les règles de construction du `Backbone` sont prioritaires par rapport à la règle $\mathbb{R}_{\texttt{Backward}}$. Si tout les ascendants $u$ de $v$ sont tel que $u \notin B(\gamma) \cup X(\gamma)$, on note $w$ le premier ascendant de $v$ ayant un enfant non visité $z$ ($w$ peut être la racine de la sous-structure couvrante). Le nombre d'activation consécutive de la règle $\mathbb{R}_{\texttt{Backward}}$ est limité par la distance entre $v$ et $z$. Le nœud $z$ n'est pas activable par la règle $\mathbb{R}_{\texttt{Backward}}$, il deviendra potentiellement activable pas la règle $\mathbb{R}_{\texttt{StartBFS}}$.

■

Notre ensemble $X(\gamma)$ est maintenant composé des nœuds activables par les règles $\mathbb{R}_{\texttt{ErToken}}$ et $\mathbb{R}_{\texttt{Backward}}$, cet ensemble peut-être vide.

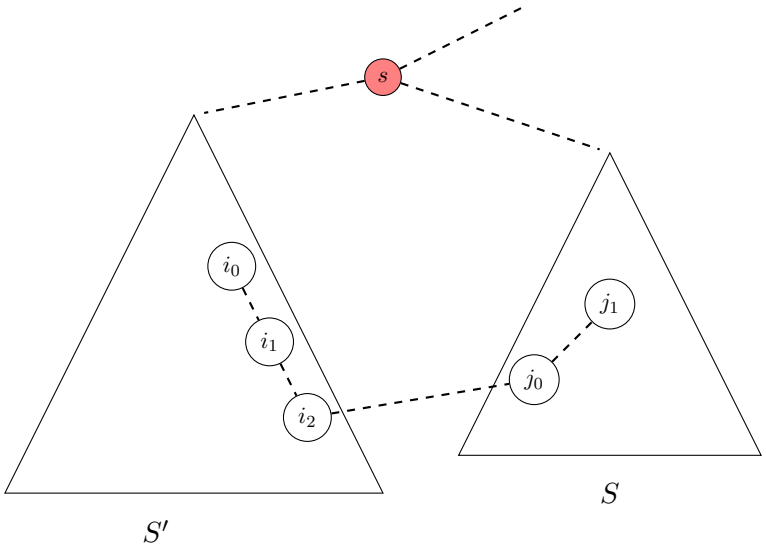



FIGURE 3.8 – Sous-structures couvrantes

Le fait que notre algorithme soit non silencieux est dû à la circulation du jeton. La circulation du jeton en descente relance le calcul du BFS. En effet l'application par un nœud $v$ ayant TokenD($j$) = *true* de la règle $\mathbb{R}_{\texttt{StartBFS}}$ change la racine du BFS ( $\mathsf{R}_{\mathsf{BFS}}$ prend l'identifiant du nœud appliquant la règle $\mathbb{R}_{\texttt{StartBFS}}$). Par la suite les autres nœuds du système vont rejoindre le BFS enraciné à cette racine se qui permettra le calcul de l'excentricité de cette racine (règles $\mathbb{R}_{\texttt{Tree}}$, $\mathbb{R}_{\texttt{BFS}}$ et $\mathbb{R}_{\texttt{Ecc}}$ ). Quand ce même nœud $v$ aura fini le calcul de son excentricité il exécutera la règle $\mathbb{R}_{\texttt{EndBFS}}$ et il changera sa variable MinEUP se qui déclenchera le calcul du centre (règles $\mathbb{R}_{\texttt{MinEUp}}$, $\mathbb{R}_{\texttt{MinERoot}}$ et $\mathbb{R}_{\texttt{MinEDown}}$). Nous allons voir que la stratégie pour le démon de faire circulé sans cesse le ou les jetons pour relancer le calcul d'excentricité et le calcul du centre est contré par notre algorithme.

Les règles de $\mathbb{R}_{\texttt{StartBFS}}$ et $\mathbb{R}_{\texttt{EndBFS}}$ sont étroitement liées. En effet, la circulation de jeton n'est possible que si un nœud active une de ces deux règles. Donc prouver que le nombre d'activation de la règle $\mathbb{R}_{\texttt{EndBFS}}$ est borné revient à prouver que le nombre d'activation de la règle $\mathbb{R}_{\texttt{StartBFS}}$ est lui aussi borné.

De plus, les règles $\mathbb{R}_{\texttt{EndBFS}}$, $\mathbb{R}_{\texttt{CleanEcc}}$, $\mathbb{R}_{\texttt{BFS}}$ et $\mathbb{R}_{\texttt{Ecc}}$ supposent toutes que le prédicat SameBFS soit vrai. Nous allons prouver que toutes les règles qui reposent sur ce prédicat ont un nombre d'activation borné.

**Propriété** 3 (SameBFS) *A partir d'une configuration $\gamma \not\subset \Gamma_{\texttt{Backbone}}$, le nombre d'activation des règles ayant le prédicat* SameBFS *est borné.*

**Preuve de la propriété.** Considérons une configuration $\gamma \not\subset \Gamma_{\texttt{Backbone}}$ et une sous-structure couvrante $S$ tel que $\forall u \in S$ on ai $u \notin B(\gamma) \cup X(\gamma)$, et un nœud $j_0 \in S$ tel que $\exists i_2 \in N_{j_0}$ avec $i_2 \in S'$ (voir Figure 3.8), avec $S'$ une autre sous-structure couvrante. Il est important de noter que ces deux sous-structures couvrantes sont peut-être dans la même sous-structure couvrante, mais dans ce cas il existe au moins un nœud $s$ entre $S$ et $S'$ tel que $s \in B(\gamma) \cup X(\gamma)$. Supposons qu'à l'instant $t$ on ai TokenD($j_0$) = *true*. Pour que la règle $\mathbb{R}_{\texttt{EndBFS}}$ qui permet la circulation du jeton en descente s'exécute il faut que le

prédicat SameBFS soit vrai. Pour rappel :

$$\text{SameBFS}(v) \equiv \{\forall u \in N_v, \mathsf{R}_{\mathsf{BFS}u} = v\}$$

Pour que le jeton continue à circuler après le temps $t$, il faut que le nœud $i_2$ ai $\mathsf{R}_{\mathsf{BFS}i_2} = j_0$ au temps $t$. Soit $j_1$ le prédécesseur de $j_0$ dans la circulation de jeton. Supposons que le démon peut faire circuler le jeton jusqu'au temps $t'$ ou $\text{TokenD}(j_1) = true$. Le nœud $j_0$ sera activable par la règle $\mathbb{R}_{\texttt{Tree}}$ et il prendra les valeurs suivantes :

$$(\mathsf{R}_{\mathsf{BFS}j_0}, \mathsf{P}_{\mathsf{BFS}j_0}, \mathsf{d}_{\mathsf{BFS}j_0}, \mathsf{D}_{\mathsf{BFS}j_0}) := (j_1, j_1, \infty, \bot)$$

1. Si $\forall w \in S'$on a $\mathsf{R}_{\mathsf{BFS}w} \notin \{j_0, j_1\}$, que le nœud $v$ soit activé ou pas il restera à $\mathsf{R}_{\mathsf{BFS}v} \neq j_1$. Pour pouvoir activer la règle $\mathbb{R}_{\texttt{EndBFS}}$, le nœud $j_0$ doit avoir $\text{SameBFS}(j_0) = true$ (autrement dit $\mathsf{R}_{\mathsf{BFS}i_2} = j_1$) en conséquence le jeton restera bloqué et le nœud $j_1$ ne sera plus activable par les règles de circulation de jeton ou de calcul d'excentricité. Il en va de même pour les autres règles reposant sur le prédicat SameBFS, puisque si il y a un voisin dans une sous-structure différente, elles ne pourront pas être activables.
2. Si $\exists w \in S'$ tel que $\mathsf{R}_{\mathsf{BFS}w} \in \{j_0, j_1\}$. Supposons que notre structure $S$ soit limitée au nœud $j_0$ et $j_1$ et que tout les autres nœuds du réseau soit dans la structure $S'$. Par hypothèse de départ on a le nœud $s$ tel que $s \in B(\gamma) \cup X(\gamma)$, sans perte de généralité supposons que $S'$ soit une chaîne de $n-2$ nœuds enraciné au nœud $s$ et ayant comme feuille le nœud $i_0$ avec $i_0 \in N_{j_0}$ (voir Figure 3.9). Supposons que l'ordonnancement des valeurs des nœuds dans $S'$ permette au démon d'activer les règles du calcul de l'excentricité tout en permettant la circulation du jeton entre $j_0$ et $j_1$ (voir Figure 3.9). A chaque circulation entre $j_0$ et $j_1$ le nombre de nœuds qui prennent la valeur du jeton augmente dans la sous-structure $S'$, donc en (au plus) $n$ circulation de jeton tout les nœuds de $S'$ aurons pris la valeur $j_0$ (ou $j_1$). Quand le jeton arrivera en $j_1$ (ou $j_0$), la valeur $j_1$ ne sera plus présente dans $S'$ ce qui rejoint le cas 1.

■

Il est important de noter que le deuxième cas de cette preuve entraine une complexité en $O(n^2)$ rondes pour cette étape.

Il nous reste maintenant à prouver que si la règle $\mathbb{R}_{\texttt{EndBFS}}$ n'est pas activée, soit parce qu'elle n'est plus activable ,soit parce que le demon ne l'active pas. L'activation des règles relatives au calcul du centre est bornée. On considère donc une configuration $\gamma$ et un ensemble $X(\gamma)$ composé de nœuds activables par la règle $\mathbb{R}_{\texttt{ErToken}}$ et par la règle $\mathbb{R}_{\texttt{Backward}}$ puisque ce sont les seuls règles plus prioritaires que celles du calcul du centre. Il est à noter que seul le nombre de nœuds activables par les règles du `Backbone` dans l'ensemble $A(\gamma)$ sont non nuls. Dans les règles de calcul du centre la plus prioritaires est la règle $\mathbb{R}_{\texttt{MinEUp}}$, nous commeçons donc par prouver que le nombre d'activation de cette règle est borné.

**Propriété** 4 ($\mathbb{R}_{\texttt{MinEUp}}$) *A partir d'une configuration $\gamma \notin \Gamma_{\texttt{Backbone}}$ le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinEUp}}$ est borné.*

**Preuve de la propriété.** Soit un nœud $v$ non activable par la règle MinEUP (l'excentricité minimum de ses enfants (variable MinE) ne change pas), $v$ peut devenir activable par la règle MinEUP si et seulement si après l'exécution de la règle $\mathbb{R}_{\texttt{EndBFS}}$ son excentricité a diminuée. Nous avons vu que le nombre d'activation de la règle $\mathbb{R}_{\texttt{EndBFS}}$ est borné, donc le

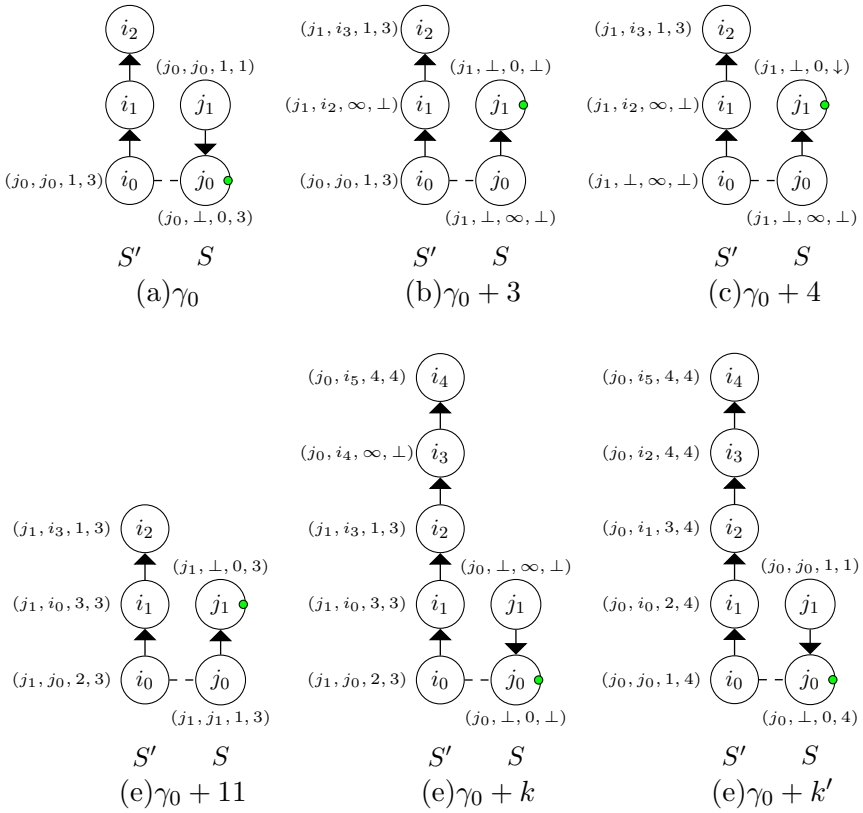



FIGURE 3.9 – Sous-structures couvrantes.
Les valeurs à côté du nœud $v$ sont les suivantes : $(\mathsf{R}_{\mathsf{BFS}_v}, \mathsf{P}_{\mathsf{BFS}_v}, \mathsf{d}_{\mathsf{BFS}_v}, \mathsf{D}_{\mathsf{BFS}_v})$

nombre de fois qu'un nœud peut devenir activable pour la règle MinEUP par l'exécution de la règle $\mathbb{R}_{\texttt{EndBFS}}$ est lui aussi borné. Soit un nœud $v$ activable par la règle MinEUP. Cette règle calcule l'excentricité minimum retournée par la fonction $\mathsf{MinEcc}(v)$ (voir 3.30), plus précisément cette fonction calcule l'excentricité minimum entre le nœud $v$ et ses enfants dans le `Backbone`. Une execution de cette règle peut déclencher un calcul ascendant de l'excentricité. Cette règle est prioritaire sur les règles de calcul du BFS et sur les règles $\mathbb{R}_{\texttt{MinERoot}}$ et $\mathbb{R}_{\texttt{MinEDown}}$. Le nombre d'activation de MinEUP lié au nœud $v$ est limité par la distance entre $v$ est le nœud $w$, avec $w$ racine de la sous-structure ou activable par les règles du `Backbone`, $\mathbb{R}_{\texttt{ErToken}}$ et $\mathbb{R}_{\texttt{Backward}}$. ■

Notons que le nombre d'activation de cette règle peut être en $O(n^2)$. Des valeurs classées en ordre décroissant sur une chaine avec une activation par le démon des nœuds au plus près de la racine pour finir avec seul les plus éloignés entraine une telle complexité.

On rajoute maintenant à l'ensemble $X(\gamma)$ les nœuds activables par la règle MinEUP. Évidement, le nombre de nœuds activables par cette règle peut-être nul dans $X(\gamma)$.

**Propriété** 5 ($\mathbb{R}_{\texttt{MinERoot}}$) *A partir d'une configuration $\gamma \not\subset \Gamma_{\texttt{Backbone}}$, le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinERoot}}$ est borné.*

**Preuve de la propriété.** Cette règle est activable uniquement par les nœuds se considérant racine du `Backbone`. Cette règle est activable à chaque fois que l'un de ses enfant lui propose une meilleure excentricité minimum, ce qui signifie qu'un de ses enfant a activé la

règle $\mathbb{R}_{\texttt{MinEUp}}$. Nous avons vu par la propriété 4 que le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinEUp}}$ est borné. Par conséquent, le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinERoot}}$ est lui aussi borné. ■

**Propriété** 6 ($\mathbb{R}_{\texttt{MinEDown}}$) *A partir d'une configuration $\gamma \not\subset \Gamma_{\texttt{Backbone}}$ le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinEDown}}$ est borné.*

**Preuve de la propriété.** La règle $\mathbb{R}_{\texttt{MinEDown}}$ sert à propager l'excentricité minimale de la racine du `Backbone` vers les feuilles. Elle est donc activable de la racine vers les feuilles à chaque fois qu'une racine (un seul nœud si le `Backbone` est construit) active la règle $\mathbb{R}_{\texttt{MinERoot}}$, nous avons vu par la propriété 5 que le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinERoot}}$ est borné. Par conséquent, le nombre d'activation de la règle $\mathbb{R}_{\texttt{MinEDown}}$ est lui aussi borné. ■

Il nous reste maintenant à traiter la dernière règle dans l'ordre des priorités, la règle $\mathbb{R}_{\texttt{Tree}}$. On rajoute maintenant dans l'ensemble $X(\gamma)$ les nœuds activables par les règles de calcul du centre qui sont plus prioritaires que $\mathbb{R}_{\texttt{Tree}}$.

**Propriété** 7 ($\mathbb{R}_{\texttt{Tree}}$) *A partir d'une configuration $\gamma \not\subset \Gamma_{\texttt{Backbone}}$ le nombre d'activation de la règle $\mathbb{R}_{\texttt{Tree}}$ est borné.*

**Preuve de la propriété.** La règle $\mathbb{R}_{\texttt{Tree}}$ s'assure qu'un nœud $v$ a la même racine de BFS que son voisin pointé par p_next($v$) (voir 3.10). Ceci est testé par ¬GoodBFS($v$)( 3.14).

$S_{TreeR}$ : La sous-structure couvrante $S_{TreeR}$ est un arbre couvrant $T_v$ enraciné en un nœud $v$. Si $T_v$ couvre l'ensemble des nœuds $v$ de $V$ alors $\Gamma \subset \Gamma_{\texttt{Backbone}}$ sinon $\Gamma \not\subset \Gamma_{\texttt{Backbone}}$. Si il existe plusieurs nœuds $j$ tel que TokenD($j$) = *true*, (autrement dit la règle $\mathbb{R}_{\texttt{ErToken}}$ est activable par des nœuds de $T_v$) chaque nœud $u \in T_v$ rejoindra le BFS de son plus proche ancêtre $j$ (induit par la fonction p_next($u$)) ayant TokenD($j$) = *true*. Il est à noter que certains nœuds auront pour ancêtre (induit par la fonction p_next($u$)) un nœud $w$ pouvant exécuter la règle $\mathbb{R}_{\texttt{ErToken}}$, dans ce cas là l'ensemble des descendant de $w$ prendront la valeur de la variable $\mathsf{d}_{\mathsf{BFS}_w}$. La diffusion de la variable $\mathsf{R}_{\mathsf{BFS}}$ s'effectue donc d'une racine induite par p_next($u$) (nœud $i$ tel que TokenD($i$) = *true* ou $i$ peut exécuter $\mathbb{R}_{\texttt{ErToken}}$) jusqu'aux feuilles, le nombre d'activation est donc borné.

$S_{TreeFR}$ : Pour la structure $S_{TreeFR}$, s'il existe plusieurs jetons dans la structure, la preuve est la même que pour $S_{TreeR}$. La seule différence est que l'on considèrera les nœuds étant activables par $\mathbb{R}_{\texttt{ErToken}}$ comme racine induite par la fonction p_next. Si il n'existe pas de jeton dans la structure, l'ensemble des nœuds prendrons la valeur de la racine de la structure, ce qui revient au cas précédent.

$S_{Cycle}$ : Pour la sous-structure couvrante $S_{Cycle}$, les nœuds du cycle (et leurs descendants) prendront la valeurs de $\mathsf{R}_{\mathsf{BFS}}$ de leur ancêtre $w$, avec $w$ qui activable par $\mathbb{R}_{\texttt{ErToken}}$ ou $w \in B$ ou enfin $w$ tel que TokenD($w$) = *true*, ce qui limite le nombre d'activations de la règle $\mathbb{R}_{\texttt{Tree}}$.Si il n'existe pas de jeton dans la structure, l'ensemble des nœuds prendrons la valeur du nœud activable par $\mathbb{R}_{\texttt{ErToken}}$ ou activable par une règle du `Backbone`, et donc en (au plus) $n$ rondes la règle $\mathbb{R}_{\texttt{Tree}}$ n'est plus activable, ce qui limite le nombre d'activation de la règle $\mathbb{R}_{\texttt{Tree}}$.

■

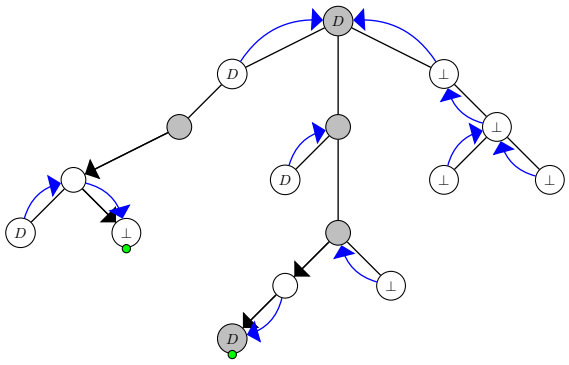



FIGURE 3.10 – Illustration de la variable p_next (flèches bleues) dans un sous-arbre couvrant avec des nœuds activables par des règles du `Backbone` ou $\mathbb{R}_{\texttt{ErToken}}$

Nous avons prouvé que le nombre d'activation des règles autres que celles de la construction du `Backbone` était borné. À partir d'une configuration initiale $\gamma_0$, le système converge vers une configuration $\gamma$ tel que $A(\gamma) \setminus B(\gamma) = \emptyset$.

□

Notons $\Gamma_{\texttt{1-Token}} \subseteq \Gamma$ l'ensemble des configurations dans lesquelles il existe exactement un jeton (c'est à dire qu'aucun nœud n'est activé par la règle $\mathbb{R}_{\texttt{ErToken}}$).

**Lemme 2** $\Gamma_{\texttt{Backbone}} \rhd \Gamma_{\texttt{1-Token}}$ *en* $O(n)$ *rondes pour* $\mathcal{SSCC}$ *sous un démon distribué non équitable.* $\Gamma_{\texttt{1-Token}}$ *est clos.*

**Preuve.** Pour la preuve de la clôture se reporter au preuves de l'algorithme [PV99] puisque nous ne modifions pas la règle $\mathbb{R}_{\texttt{ErToken}}$. La preuve du lemme 1 est directement utilisable pour la convergence. En effet, nous avons toujours considéré deux ensembles, l'ensemble $B$ contenant les nœuds activables par les règles de construction du `Backbone` et l'ensemble $X$ contenant les nœuds activables que le démon ne voulait pas activé. A partir de la preuve de la propriété 2 nous avons considéré que le démon ne souhaitait pas activer les nœuds activables par $\mathbb{R}_{\texttt{ErToken}}$. Chacune des preuves suivantes considère des structures couvrantes, structures qui peuvent couvrir l'ensemble du réseau, hors c'est le cas ici puisque le `Backbone` est construit. □

Définissons par $E^*$ l'excentricité minimale du réseau et $\Gamma_{\texttt{Centers}}$ l'ensemble des configurations tel que pout tout $v$ dans $V$ et $\mathsf{MinE}_v = E^*$. Les centres du réseau sont les nœuds avec $\mathsf{Ecc}_v = \mathsf{MinE}_v$.

**Lemme 3** $\Gamma_{\texttt{1-Token}} \rhd \Gamma_{\texttt{Centers}}$ *en* $O(n^2)$ *rondes pour* $\mathcal{SSCC}$ *sous un démon distribué non équitable.* $\Gamma_{\texttt{Centers}}$ *est clos.*

**Preuve.** Nous commeçons notre preuve en prouvant qu'en la présence d'un `Backbone` et d'un unique jeton, le jeton n'est pas bloqué indéfiniment.

**Définition 27** ***chemin bloquant*** *: Pour chaque nœud $v$, on définit un* chemin bloquant *de $v$ comme un chemin maximal dans le $\texttt{BFS}(v)$ à partir de $v$ vers un autre nœud du $\texttt{BFS}(v)$ tel que chaque nœud $u$ sur ce chemin satisfasse $\mathsf{D}_{\mathsf{BFS}_u} \notin \{\perp, \downarrow, \uparrow\}$.*

Soit $\lambda$ la fonction de potentiel telle que $\Gamma_{\texttt{1-Token}} \times V \to \mathbb{N}$ où la fonction $\lambda$ est définit de la façon suivante :

$$\lambda(\gamma, r, v) \quad = \quad \begin{cases} 1 & \text{si } v \text{ appartient au chemin bloquant de } r \text{ dans } \gamma \\ 1 & \text{si } v \notin \text{SameBFS}(r) \text{ et } v \in N_r \text{ dans } \gamma \\ 0 & \text{sinon} \end{cases}$$

Maintenant, nous définissons notre fonction de potentiel $\Lambda : \Gamma_{\texttt{1-Token}} \to \mathbb{N}$ comme suit :

$$\Lambda(\gamma, r) = \sum_{v \in V} \lambda(\gamma, r, v)$$

**Propriété** 8 *$\Lambda(\gamma, r) > 0$ implique que le jeton est bloqué.*

**Preuve de la propriété.** Le jeton est relâché lorsque le nœud $r$ exécute la règle $\mathbb{R}_{\texttt{EndBFS}}(3.29)$, mais cette règle est activable seulement si tout les voisins de $r$ sont dans $\texttt{BFS}(r)$ (voir prédicat SameBFS$(v)$( 3.16)) et tout les enfants de $r$ (autrement dit tout les voisins de $r$) ont déjà calculé la distance maximal de leur sous-arbre (voir prédicat ChN$(v)$ (3.28) dans la règle $\mathbb{R}_{\texttt{EndBFS}}$). Si ce n'est pas le cas, alors le jeton est bloqué et par définition de $\Lambda$, nous avons $\Lambda(\gamma, r) > 0$. ■

**Propriété** 9 *Si $\Lambda(\gamma, r) > 0$, alors l'activation de $\mathbb{R}_{\texttt{Tree}}$ par un nœud $v \in V$ dans la configuration $\gamma$ donne une configuration $\gamma'$ telle que $\Lambda(\gamma', r) = \Lambda(\gamma, r)$.*

**Preuve de la propriété.** Considérons une configuration $\gamma$ avec $\Lambda(\gamma, r) > 0$ et $A(\gamma)$ l'ensemble des nœuds activés par la règle $\mathbb{R}_{\texttt{Tree}}(3.13)$. Plus précisément, la règle $\mathbb{R}_{\texttt{Tree}}(3.13)$ est activable quand le prédicat GoodBFS$(v)$ ( 3.14) n'est pas vrai, autrement dit quand $\mathsf{R}_{\mathsf{BFS}_v} \neq \mathsf{R}_{\mathsf{BFS}_{\text{p\_next}_v}}$. Rappelons que $\text{p\_next}_v$ est une fonction qui retourne l'identifiant du voisin de $v$ qui appartient au chemin du `Backbone` de $v$ à $r$ (voir 3.10). On distingue deux cas :

1. Considérons $v$ un voisin de $r$, si $\mathbb{R}_{\texttt{Tree}}$ est activable par $v$, ce qui veut dire que $v \notin \text{SameBFS}(r)$ alors $\lambda(\gamma, r, v) = 1$, l'exécution de la règle $\mathbb{R}_{\texttt{Tree}}$ par $v$ dans $\gamma$ donne $\mathsf{D}_{\mathsf{BFS}_v} = \perp$ dans $\gamma'$, ceci maintient $\lambda(\gamma', r, v) = 1$.
2. Considérons maintenant $v$ non voisin de $r$, si $\mathbb{R}_{\texttt{Tree}}$ est activable par $v$, cela signifie que $\mathsf{R}_{\mathsf{BFS}_v} \neq \mathsf{R}_{\mathsf{BFS}_{\text{p\_next}_v}}$, alors soit $\mathsf{R}_{\mathsf{BFS}_{\text{p\_next}_v}} \neq r$ et par conséquent $\lambda(\gamma, r, v) = \lambda(\gamma', r, v) = 0$, soit $\mathsf{R}_{\mathsf{BFS}_{\text{p\_next}_v}} = r$ alors $\lambda(\gamma, r, v) = 0$ et $\lambda(\gamma', r, v) = 0$. Comme l'exécution de la règle $\mathbb{R}_{\texttt{Tree}}$ par $v$ dans $\gamma$ donne $\mathsf{P}_{\mathsf{BFS}_v} = \perp$ ($v$ n'est pas dans $\texttt{BFS}(v)$ dans $\gamma'$ puisque $v$ n'a pas de parent).

Comme conséquence directe des deux cas, nous obtenons $\Lambda(\gamma', r) = \Lambda(\gamma, r)$. ■

Définissons $\Gamma_{\texttt{Tree(r)}} \subseteq \Gamma_{\texttt{1-Token}}$ comme l'ensemble des configurations $\gamma \in \Gamma_{\texttt{1-Token}}$ tel que la règle $\mathbb{R}_{\texttt{Tree}}$ n'est activable par aucun nœud et $\Lambda(\gamma, r) > 0$. D'après la propriété 7 le nombre d'activation de la règle $\mathbb{R}_{\texttt{Tree}}$ est borné.

**Propriété** 10 *Si $\Lambda(\gamma, r) > 0$, alors l'activation de $\mathbb{R}_{\texttt{BFS}}$ par n'importe quel nœud $v \in V$ dans la configuration $\gamma$ donne une configuration $\gamma'$ tel que $\Lambda(\gamma', r) \geq \Lambda(\gamma, r)$.*

**Preuve de la propriété.** L'activation de la règle $\mathbb{R}_{\texttt{BFS}}$( 3.15) par le nœud $v$ donne $\mathsf{D}_{\texttt{BFS}v} = \downarrow$. En conséquence, si $v$ était dans un chemin bloquant, $v$ reste dans un chemin bloquant. D'autre part, si $v$ était un voisin d'un nœud $u$ élément d'un chemin bloquant, le nœud $v$ rejoint le chemin bloquant et $\Lambda(\gamma', r)$ augmente d'au moins un comparé à $\Lambda(\gamma, r)$. ■

Définissons $\Gamma_{\texttt{BFS}}$ une configuration $\gamma$ où $\forall v \in V$, $\mathsf{d}_{\texttt{BFS}v} = d_G(r, v)$ et $\Lambda(\gamma) > 0$. En d'autres termes, la règle $\mathbb{R}_{\texttt{BFS}}$ n'est activable par aucun nœud.

**Propriété** 11 *À partir d'une configuration $\gamma \in \Gamma_{\texttt{Tree(r)}}$ avec $\Lambda(\gamma, r) > 0$, le nombre d'activations de $\mathbb{R}_{\texttt{BFS}}$ est borné.*

Le calcul de l'arbre BFS avec une seule règle est bien connu mais à notre connaissance, aucune preuve n'utilise de fonction de potentiel. Pour être cohérent, nous proposons une nouvelle preuve basée sur une fonction de potentiel. Les auteurs de [HC92] ont proposé une fonction de potentiel mais celle-ci ne convient pas dans notre cas (en raison d'une caractérisation trop faible des effets de l'algorithme sur les configurations et de l'utilisation d'une *a priori* connaissance des nœuds). De plus, l'algorithme dans [HC92] utilise 3 règles et la preuve repose sur le parent de chaque nœud. Nous généralisons cette preuve pour notre algorithme qui utilise une seule règle pour la construction dun arbre couvrant BFS.

**Preuve de la propriété.** Soit $\gamma$ une configuration dans $\Gamma_{\texttt{Tree(r)}}$ et $A(\gamma)$ l'ensemble de tout les nœuds activés par la règle $\mathbb{R}_{\texttt{BFS}}$ dans $\gamma$. Après l'activation des nœuds de $A(\gamma)$, le système atteint la configuration $\gamma'$. Notez que seule la règle $\mathbb{R}_{\texttt{BFS}}$ peut changer la distance dans l'arbre BFS (en d'autres termes la variable $\mathsf{d}_{\texttt{BFS}}$). Afin de clarifier notre résultat, on note $\mathsf{d}_{\texttt{BFS}v}(\gamma)$ la distance dans l'arbre BFS du nœud $v$ dans la configuration $\gamma$. Pour calculer un BFS, un nœud $v$ attend que tous ses voisins partagent la même racine BFS que lui (voir SameBFS 3.16). En outre, la règle $\mathbb{R}_{\texttt{BFS}}$ est activable uniquement lorsque la distance du nœud $v$ peut être améliorée (voir Best 3.17). Pour convenance, nous introduisons $\mathsf{min}^{+1}(v, \gamma) = \min\{\mathsf{d}_{\texttt{BFS}u}(\gamma) \mid u \in N_v\} + 1$ qui est le résultat de la fonction Best$(v)$. Soit $P(\gamma)$ la fonction de potentiel suivante :

$$P(\gamma) = (F(\gamma), S(\gamma))$$

où

$$S(\gamma) = \sum_{v \in V} \mathsf{d}_{\texttt{BFS}v}(\gamma)$$

et

$$F(\gamma) = (f_1(\gamma), f_2(\gamma), \ldots, f_{2n+\max\{\mathsf{d}_{\texttt{BFS}v}(\gamma_0) | v \in V\}}(\gamma))$$

où $\mathtt{f}_k(\gamma) = \{v | \mathsf{d}_{\texttt{BFS}v}(\gamma) = k \wedge \mathsf{d}_{\texttt{BFS}v}(\gamma) \leq \mathsf{min}^{+1}(v, \gamma)\}$ et $f_k(\gamma) = |\mathtt{f}_k(\gamma)|$. Notons $K$ l'ensemble $\{1, \ldots, 2n + \max\{\mathsf{d}_{\texttt{BFS}v}(\gamma_0) | v \in V\}\}$. Notez que, par définition, tout les nœuds $v$ en $\mathtt{f}_k(\gamma)$ pour tout $k \in K$ sont activables par la règle $\mathbb{R}_{\texttt{BFS}}$. Désignons par $m$ la distance telle que $m = \min\{k \mid k \in K \wedge \exists v \in A(\gamma)$ s.t $v \in \mathtt{f}_k(\gamma)\}$. La comparaison entre $F(\gamma)$ et $F(\gamma')$ se fait par ordre lexical. On note $\gamma'$ la configuration après l'activation des nœuds $A(\gamma)$. Nous pouvons maintenant démontrer le résultat suivant $P(\gamma') < P(\gamma)$ pour toute configuration $\gamma$ et $A(\gamma)$ non vide.

1. $\exists v \in V$ tel que $v \notin \mathtt{f}_k(\gamma), \forall k \in K$. La seule façon pour un nœud $v$ de devenir élément de $\mathtt{f}_s(\gamma')$ est d'avoir un voisin $u = \mathsf{min}^{+1}(v, \gamma')$ tel que $u \in \mathtt{f}_t(\gamma)$ et $u \in A(\gamma)$. Soit $w$ le voisin de $u$ tel que $w = \mathsf{min}^{+1}(u, \gamma)$ et supposons que $t = m$. Par définition des nœuds $u, v, w$, nous avons $\mathsf{d}_{\mathsf{BFS}_v}(\gamma) > m \leq \mathsf{d}_{\mathsf{BFS}_w}(\gamma)$. Après l'activation du nœud $u$, on obtient $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') \leq \mathsf{d}_{\mathsf{BFS}_u}(\gamma') = \mathsf{dfs}_w(\gamma) + 1$. Si $v \notin A(\gamma)$, nous avons $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') = \mathsf{d}_{\mathsf{BFS}_v}(\gamma)$ sinon si $v \in A(\gamma)$ après l'activation de $v$, nous obtenons que $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') = m + 1$ alors $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') \geq m + 1$ et par conséquent $f_m$ est diminuée d'au moins un dans $\gamma'$ et le système converge vers $F(\gamma') < F(\gamma)$.
2. $\exists v \in A(\gamma)$ tel que $v \in \mathtt{f}_k(\gamma)$, alors $\mathsf{d}_{\mathsf{BFS}_v}(\gamma) \leq \mathsf{d}_{\mathsf{BFS}_u}(\gamma)$ avec $u = \mathsf{min}^{+1}(v, \gamma)$ et après l'activation de $v$, on obtient $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') = \mathsf{d}_{\mathsf{BFS}_u}(\gamma') + 1$. Sans perte de généralité, supposons que $k = m$ et considérons $w$ le nœud tel que $w = \mathsf{min}^{+1}(v, \gamma')$.
   (a) Prouvons maintenant que si $u = w$, nous obtenons $F(\gamma') < F(\gamma')$ :
      i. Si $u \notin A(\gamma)$, nous obtenons $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') > \mathsf{d}_{\mathsf{BFS}_u}(\gamma')$, alors $v \notin f_s(\gamma') \forall s \in K$. Par conséquent, $f_m$ décroit d'au moins un.
      ii. $u \in A(\gamma)$ s.t. $u \in \mathtt{f}_\mathtt{s}(\gamma) : m \leq s \leq \mathsf{d}_{\mathsf{BFS}_z}(\gamma)$ avec $z = \mathsf{min}^{+1}(u, \gamma)$, après activation de $v$ et $u$ nous obtenons : $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') = s+1 \leq \mathsf{d}_{\mathsf{BFS}_u}(\gamma') = \mathsf{d}_{\mathsf{BFS}_z}(\gamma)+1$ alors $v \in \mathtt{f}_{\mathtt{s}+1}(\gamma')$ en conséquence $f_m$ décroit d'au moins un.
      iii. $u \in A(\gamma)$ et $u \notin \mathtt{f}_s(\gamma) \forall s \in K$ alors $\mathsf{d}_{\mathsf{BFS}_u}(\gamma) > \mathsf{d}_{\mathsf{BFS}_z}(\gamma)$ avec $z = \mathsf{min}^{+1}(u, \gamma)$. Après l'activation de $v$ et $u$, nous obtenons $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') = \mathsf{d}_{\mathsf{BFS}_u}(\gamma) + 1 > \mathsf{d}_{\mathsf{BFS}_u}(\gamma') = \mathsf{d}_{\mathsf{BFS}_z}(\gamma) + 1$, alors $v \notin \mathtt{f}_s(\gamma') \forall s \in K$. En conséquence, $f_m$ décroit d'au moins un.
   (b) Prouvons maintenant que si $u \neq w$, nous obtenons $F(\gamma') < F(\gamma')$ :
      i. $u \notin A(\gamma)$ et $w \notin A(\gamma)$ : ceci est impossible parce que $u \neq w$.
      ii. $u \in A(\gamma)$ et $w \notin A(\gamma)$ où $u \in A(\gamma)$ et $w \in A(\gamma)$ : par définition de $u$ et $w$ nous avons $\mathsf{d}_{\mathsf{BFS}_u}(\gamma) = m \leq \mathsf{d}_{\mathsf{BFS}_w}(\gamma)$ et nous obtenons, après l'activation des nœuds, que $\mathsf{d}_{\mathsf{BFS}_v}(\gamma') = m + 1 \leq \mathsf{d}_{\mathsf{BFS}_u}(\gamma') = \mathsf{d}_{\mathsf{BFS}_w}(\gamma') + 1$. Alors $f_m$ décroit d'au moins un.

   Pour conclure cette partie : si $\exists v \in A(\gamma)$ tel que $v \in \mathtt{f}_k(\gamma)$, la fonction $F$ décroit et par conséquent $P(\gamma') < P(\gamma)$.
3. $\forall v \in A(\gamma) : v \notin \mathtt{f}_k(\gamma), \forall k \in K$.
   Une conséquence directe du premier élément de cette preuve est que l'activation des nœuds maintient $F(\gamma') = F(\gamma)$. En outre, cette activation diminue la distance de $v$ alors $S(\gamma') < S(\gamma)$ et $P(\gamma') < P(\gamma)$.

■

*Corollaire* 1 *À partir d'une configuration $\gamma \in \Gamma_{\mathtt{Tree(r)}}$, le système converge en $O(n)$ rondes dans les configurations $\gamma' \in \Gamma_{\mathtt{BFS}}$.*

Soit $A(\gamma)$ l'ensemble des nœuds activables dans $\gamma$ et $\mathsf{notCoh}(\gamma) \subseteq A(\gamma)$ un ensemble de nœuds tel que si $v$ dans $\mathsf{notCoh}(\gamma)$ implique $\mathsf{CohD}(v) = false$ (voir 3.23, 3.21 et 3.22).

**Propriété** 12 *À partir d'une configuration $\gamma \in \Gamma_{\mathtt{BFS}}$ avec $\Lambda(\gamma, r) > 0$ et si le démon n'active jamais les éléments* notCoh *et ne décroit jamais $\Lambda$, alors le système converge vers une configuration $\gamma'$ tel que* $\mathsf{notCoh}(\gamma') = A(\gamma')$ .

**Preuve de la propriété.** Sans perte de généralité, nous pouvons réduire notre étude à trois cas pour un nœud $v$ tel que $v \in \mathsf{notCoh}(\gamma)$ :

1. tout les descendants $u$ de $v$ dans $\mathtt{BFS}(r)$ ont $\mathsf{CohD}(u) = true$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \uparrow$, alors tout les nœuds $u$ ont $\mathsf{D}_{\mathsf{BFS}_u} = \uparrow$ et ne sont pas activables jusqu'à ce que $v$ soit activé. En d'autres termes, pour tout $u$ descendant de $v$, nous avons $u \notin A(\gamma)$. Nous précisons que la notion de descendant est donnée par le prédicat Ch_bfs (voir 3.19).
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \perp$, notons $w$ le premier descendant dans chaque branche du sous-arbre $\mathtt{BFS}$ de $v$ tel que $\mathsf{D}_{\mathsf{BFS}_w} = \uparrow$, alors tout les descendants entre $v$ et $w$ seront à $\perp$ et tout les descendants de $w$ à $\uparrow$. Les descendants à $\uparrow$ seront activable de façon descendante pour changer leur état à $\perp$ quand ils auront tous rejoint $\perp$ on obtiendra $\forall u \notin A(\gamma)$ jusqu'à ce que $v$ soit activé.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \downarrow$, alors la configuration qui maximise le nombre d'activations pour les descendants dans le $\mathtt{BFS}(r)$ de $v$ est la suivante. Tout nœud $u$ a $\mathsf{D}_{\mathsf{BFS}_u} = \perp$ dans la vague ascendante en utilisant la règle $\mathbb{R}_{\mathtt{Ecc}}$ (voir 3.18), tout les descendants mettent leur variables $\mathsf{D}_{\mathsf{BFS}_u}$ à $\downarrow$ et on obtient une configuration $\gamma'$ où pour tout $u$, $u \notin E(\gamma')$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} \in \mathbb{N}$, alors les nœuds $u$ ont $\mathsf{D}_{\mathsf{BFS}_u} \in \mathbb{N}$ et ne sont pas activables jusqu'à ce que $v$ soit activé. En d'autres termes, pour tout $u$ descendant de $v$, nous avons $\forall u \notin A(\gamma)$.
2. tout les ancêtres $u$ dans $\mathtt{BFS}(r)$ de $v$ ont $\mathsf{CohD}(u) = true$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \uparrow$, alors la configuration qui maximise le nombre d'activations pour les $\mathtt{BFS}(r)$ ancêtres de $v$ est la suivante, tout les nœuds $u$ ont $\mathsf{D}_{\mathsf{BFS}_u} \neq \uparrow$. Dans ce contexte, dans une vague ascendante, tout les ancêtres mettent leurs variables $\mathsf{D}_{\mathsf{BFS}_u} = \uparrow$ (règle $\mathbb{R}_{\mathtt{Ecc}}$), suivie par une vague descendante à partir de la racine où tout les ancêtres mettent leurs variables $\mathsf{D}_{\mathsf{BFS}_u} = \perp$, suivie de $\mathsf{D}_{\mathsf{BFS}_u} = \downarrow$. Maintenant, jusqu'à l'activation de $v$ les ancêtres de $v$ ne sont plus activables, à savoir $\forall u \notin A(\gamma')$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \perp$, alors on doit considérer deux cas. Soit tout les ancêtres $u$ de $v$ sont $\mathsf{D}_{\mathsf{BFS}_v} = \perp$, soit il existe un ancêtre $w$ tel que tout les ancêtres de $w$ et $w$ lui même ont $\mathsf{D}_{\mathsf{BFS}} = \downarrow$ et tout les ancêtres de $v$ descendants de $w$ ont $\mathsf{D}_{\mathsf{BFS}_v} = \perp$. Quand $w$ sera le parent de $v$ l'ensemble des ancêtres de $v$ ne seront pas activables tant que $v$ n'aura pas été activé, autrement dit $\forall u \notin A(\gamma)$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \downarrow$, alors tout les $\mathtt{BFS}(r)$ ancêtres $u$ de $v$ ont $\mathsf{D}_{\mathsf{BFS}_v} = \downarrow$, et jusqu'à l'activation de $v$, nous avons $u \notin A(\gamma)$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} \in \mathbb{N}$, alors tout les ancêtres $u$ dans $\mathtt{BFS}(r)$ de $v$ ont $\mathsf{D}_{\mathsf{BFS}_u} \in \{\downarrow\} \cup \mathbb{N}$, dans le cas où les ancêtres sont $\downarrow$ il vont calculer de façon ascendante leur distance maximum et on obtiendra $\mathsf{D}_{\mathsf{BFS}_u} \in \mathbb{N}$ et jusqu'à l'activation de $v$, nous avons $u \notin A(\gamma)$.
3. Soit $u$ le premier ancêtre de $v$ tel que $u \in \mathsf{notCoh}(\gamma)$. Notons par $w$ les ancêtres de $v$, descendants de $u$, il va de soi que quelque soit $w$ on a $w \notin \mathsf{notCoh}(\gamma)$ .
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \uparrow$, alors tout les nœuds $w$ ont $\mathsf{D}_{\mathsf{BFS}_w} = \uparrow$ et jusqu'à l'activation de $v$, les nœuds $w$ restent non activables, alors $\forall w \notin A(\gamma)$.
   - Si $\mathsf{D}_{\mathsf{BFS}_v} = \perp$, alors on note $w'$ l'ancêtre de $v$ descendant de $u$ tel que $w'$ soit le premier ancêtre de $v$ avec $\mathsf{D}_{\mathsf{BFS}_{w'}} = \downarrow$. Tout les ancêtres compris entre $v$ et $w'$ auront $\mathsf{D}_{\mathsf{BFS}} = \perp$, et tous ceux entre $w'$ et $u$ auront $\downarrow$. Le système convergera vers une configuration où $w'$ sera le parent de $v$ dans le $\mathtt{BFS}$, à ce moment là on obtiendra $\forall w \notin A(\gamma)$.

- Si $\mathsf{D}_{\mathtt{BFS}_v} = \downarrow$, alors tout les nœuds $w$ ont $\mathsf{D}_{\mathtt{BFS}_w} = \downarrow$ et $\forall w \notin A(\gamma)$.
- Si $\mathsf{D}_{\mathtt{BFS}_v} \in \mathbb{N}$, alors tout les ancêtres $w$ dans $\mathtt{BFS}(r)$ de $v$ ont $\mathsf{D}_{\mathtt{BFS}_w} \in \{\downarrow\} \cup \mathbb{N}$, dans le cas où les ancêtres sont $\downarrow$ il vont calculer de façon ascendante leur distance maximum et on obtiendra $\mathsf{D}_{\mathtt{BFS}_u} \in \mathbb{N}$ et jusqu'à l'activation de $v$, nous avons $u \notin A(\gamma)$.

Il est important de remarquer que durant tout ce processus, les nœuds activables dans $\mathsf{notCoh}$ sont activables mais non activés, c'est pour cette raison que ce processus ne prend aucune ronde. ■

Soit $\Gamma_{\mathtt{GoodE}}$ un ensemble de configurations $\gamma \in \Gamma_{\mathtt{BFS}}$ où $\forall v \in V$, $\mathsf{CohD}(v) = true$.

**Propriété** 13 *À partir d'une configuration $\gamma \in \Gamma_{\mathtt{BFS}}$ avec $\Lambda(\gamma, r) > 0$ et $\mathsf{notCoh}(\gamma) = \mathsf{A}(\gamma)$, le système converge vers une configuration $\gamma' \in \Gamma_{\mathtt{GoodE}}$.*

**Preuve de la propriété.** Notre algorithme utilise des vagues ascendantes et descendantes. Si le démon veut ralentir la convergence, il active les nœuds $v$ avec $\mathsf{CohD}(v) = false$ un par un et ceci à partir des feuilles de $\mathtt{BFS}(r)$ vers la racine de $\mathtt{BFS}(r)$. Nous avons déjà vu dans la preuve de la propriété 12 que le système va rejoindre une configuration où l'ensemble des nœuds activables va être réduit aux nœuds $v$ avec $\mathsf{CohD}(v) = false$. Quand ces nœuds seront enfin activés, nous avons vu dans la même preuve que le système rejoindra une configuration où tout les nœuds seront cohérents. Nous faisons remarquer que durant tout ce processus, le démon peut maintenir $\Lambda$ positif. ■

Pour établir la propriété, nous considérons $\phi : \Gamma_{\mathtt{GoodE}} \to \mathbb{N}$ une fonction de potentiel tel que :

$$\phi(\gamma, v, r) \quad : \quad \begin{cases} n^2 & \text{si } \mathsf{D}_{\mathtt{BFS}_v} = \perp \\ n & \text{si } \mathsf{D}_{\mathtt{BFS}_v} = \uparrow \\ 1 & \text{si } \mathsf{D}_{\mathtt{BFS}_v} = \downarrow \\ 0 & \text{si } \mathsf{D}_{\mathtt{BFS}_v} \in \mathbb{N} \end{cases}$$

et nous définissions la fonction de potentiel : $\Phi : \Gamma_{\mathtt{GoodE}} \to \mathbb{N}$ comme suit :

$$\Phi(\gamma, r) = \sum_{v \in V \setminus \{r\}} \mu(\gamma, v, r)$$

**Propriété** 14 *Soit $\gamma \in \Gamma_{\mathtt{GoodE}}$ une configuration où $\Lambda(\gamma, r) > 0$, alors l'activation de $\mathbb{R}_{\mathtt{Ecc}}$ par un nœud $v \in V$ dans la configuration $\gamma$ donne une configuration $\gamma'$ tel que $\Phi(\gamma, r) > \Phi(\gamma', r)$.*

La preuve est directe par définition de la règle $\mathbb{R}_{\mathtt{Ecc}}$ et définition de $\Phi$.

**Propriété** 15 *Soit $\gamma \in \Gamma_{\mathtt{GoodE}}$ une configuration où $\Lambda(\gamma, r) > 0$ $\Phi(\gamma, r) < n$, alors l'activation de $\mathbb{R}_{\mathtt{Ecc}}$ par un nœud $v \in V$ dans la configuration $\gamma$ donne une configuration $\gamma'$ tel que $\Lambda(\gamma, r) > \Lambda(\gamma', r)$.*

**Preuve de la propriété.** Dans une configuration $\gamma \in \Gamma_{\mathtt{GoodE}}$, si $\Phi(\gamma, r) < n$, alors tout nœud $v$ a soit $\mathsf{D}_{\mathtt{BFS}_v} = \downarrow$ ou $\mathsf{D}_{\mathtt{BFS}_v} \in \mathbb{N}$. De plus, si $\Lambda(\gamma, r) > 0$, alors il existe un

chemin bloquant composé uniquement par la valeur $\downarrow$, et uniquement les feuilles de ce chemin bloquant sont activables. Alors, à chaque activation d'un nœud $v$ avec $\mathsf{D}_{\mathsf{BFS}_v} = \downarrow$, la fonction $\Lambda$ décroit de 1. ■

Définissons $\Gamma_{\mathtt{e}}$ un ensemble de configurations $\gamma \in \Gamma_{\mathtt{BFS}}$ où chaque nœud $v \in V$ a $\mathsf{D}_{\mathsf{BFS}_v} = \max\{\mathsf{d}_{\mathsf{BFS}_v}, \max\{\mathsf{D}_{\mathsf{BFS}_u} \mid u \in \mathrm{Ch\_bfs}(v)\}\}$.

*Corollaire* 2 $\Gamma_{\mathtt{BFS}} \rhd \Gamma_{\mathtt{e}}$ *en* $O(n)$ *rondes.*

**Propriété** 16 *Le jeton n'est pas bloqué indéfiniment.*

**Preuve de la propriété.** Supposons que le démon peut bloquer le jeton indéfiniment sur le nœud $r$. En d'autres termes, à partir d'une configuration $\gamma_0$ avec $\Lambda(\gamma_0, v) > 0$, le démon maintient pour toutes les configurations $\gamma > \gamma_0$ tel que $\Lambda(\gamma, v) > 0$. Nous avons vu dans la propriété 9 l'utilisation de la règle $\mathbb{R}_{\mathtt{Tree}}$. La règle $\mathbb{R}_{\mathtt{Tree}}$ ne peut décroitre $\Lambda$ mais le démon ne peut pas activer indéfiniment la règle $\mathbb{R}_{\mathtt{Tree}}$ (voir propriété 7). De façon similaire, nous avons vu dans la propriété 10 que l'utilisation de la règle $\mathbb{R}_{\mathtt{BFS}}$ ne peut pas diminuer $\Lambda$ mais que le démon ne peut pas activer indéfiniment la règle $\mathbb{R}_{\mathtt{Tree}}$ (voir propriété 11). La dernière règle disponible si le jeton est bloqué est la règle $\mathbb{R}_{\mathtt{Ecc}}$. La preuve de la propriété 12 prouve que le démon ne peut pas maintenir indéfiniment les nœuds avec une erreur dans la variable $\mathsf{D}_{\mathsf{BFS}}$ (voir prédicat $\mathsf{CohD}$). En conséquence, le système converge vers des configurations sans erreurs de la variable $\mathsf{D}_{\mathsf{BFS}}$. Pour conclure, les propriétés 13, 14 et 15 prouvent que l'application de la règle $\mathbb{R}_{\mathtt{Ecc}}$ diminue la fonction $\Lambda$ et par conséquent, le nœud $r$ devient le seul nœud activable, et l'activation de la règle $\mathbb{R}_{\mathtt{EndBFS}}$ par $r$ libère le jeton.

■

Définissons maintenant $\Gamma_{\mathtt{TokDown}} \subseteq \Gamma_{\mathtt{1-Token}}$ l'ensemble des configurations où la circulation du jeton est descendante et $\Gamma_{\mathtt{TokUp}} \subseteq \Gamma_{\mathtt{1-Token}}$ l'ensemble des configurations où la circulation du jeton est ascendante. Une conséquence directe de la propriété 16 est la suivante :

*Corollaire* 3 $\Gamma_{\mathtt{1-Token}} \rhd \Gamma_{\mathtt{TokDown}} \cup \Gamma_{\mathtt{TokUp}}$ *en* $O(n)$ *rondes et* $\Gamma_{\mathtt{TokDown}} \cup \Gamma_{\mathtt{TokUp}}$ *est clos sous* $\mathcal{SSCC}$.

Lorsque le jeton atteint un nœud $r$ dans une configuration $\gamma \in \Gamma_{\mathtt{TokDown}}$, cela implique que $\Lambda(\gamma, r) > 0$. Par la propriété 7 et la propriété 11, nous obtenons :

*Corollaire* 4 $\Gamma_{\mathtt{TokDown}} \rhd \Gamma_{\mathtt{Tree(r)}}$ *en* $O(n)$ *rondes et* $\Lambda$ *reste positif.*

*Corollaire* 5 $\Gamma_{\mathtt{Tree(r)}} \rhd \Gamma_{\mathtt{BFS}}$ *en* $O(n)$ *rondes et* $\Lambda$ *reste positif.*

Par le corollaire 2 et la propriété 16, nous obtenons :

**Propriété** 17 $\Gamma_{\mathtt{BFS}} \rhd \Gamma_{\mathtt{e}}$ *en* $O(n)$ *rondes.*

**Preuve de la propriété.**

Nous avons montré que tant que un nœud est activable par la règle $\mathbb{R}_{\texttt{BFS}}$, il existe un chemin bloquant vers le nœud $r$ possédant le jeton (voir propriété 10). Soit $w$ le dernier nœud à activer la règle $\mathbb{R}_{\texttt{BFS}}$, nous avons vu par la propriété 12 que tout les nœuds $z$ du système vont rejoindre une configuration tel que $\mathsf{D}_{\texttt{BFS}} = \downarrow$. Par la suite, la règle $\mathbb{R}_{\texttt{Ecc}}$ va être activée pour tout les nœuds des feuilles vers la racine. Les distances à la racine du $\texttt{BFS}(r)$ étant exactes (voir preuve de la propriété 11), la règle $\mathbb{R}_{\texttt{Ecc}}$ va affecter à la variable $\mathsf{D}_{\texttt{BFS}}$ la valeur de la fonction $\mathsf{MaxD}(v)$ (voir 3.20). Donc quand le nœud $r$ exécutera la règle $\mathbb{R}_{\texttt{EndBFS}}$, sa variable $\mathsf{Ecc}_r$ prendra la valeur $\max\{\mathsf{d}_{\texttt{BFS}v}, v \in V\}$ qui est l'excentricité du nœud. Autrement dit $\mathsf{Ecc}_r = e^*(r)$ avec $e^*(r)$ l'excentricité de $r$.

■

**Propriété** 18 $\Gamma_{\texttt{e}} \rhd \Gamma_{\texttt{TokDown}} \cup \Gamma_{\texttt{TokUp}}$ *en une ronde.*

**Preuve de la propriété.** Dans une configuration $\gamma \in \Gamma_{\texttt{e}}$, seul le nœud $r$ est activable par la règle $\mathbb{R}_{\texttt{EndBFS}}$. Après l'activation du nœud $r$, $\mathsf{Ecc}_r = e^*(r)$ où $e^*(r)$ est l'excentricité du nœud $r$, le jeton est libéré. ■

Soit $\Gamma_{\texttt{Ecc}} \subseteq \Gamma_{\texttt{TokDown}} \cup \Gamma_{\texttt{TokUp}}$ l'ensemble des configurations $\gamma \in \Gamma_{\texttt{r}}$ où chaque node $v \in V$ a $\mathsf{Ecc}_v = e^*(v)$, où $e^*(v)$ est l'excentricité du nœud $v$.

**Propriété** 19 $\Gamma_{\texttt{TokDown}} \cup \Gamma_{\texttt{TokUp}} \rhd \Gamma_{\texttt{Ecc}}$ *en* $O(n^2)$ *rondes, et* $\Gamma_{\texttt{Ecc}}$ *est clos sous* $\mathcal{SSCC}$.

**Preuve de la propriété.** À partir d'une configuration $\gamma \in \Gamma_{\texttt{TokDown}} \cup \Gamma_{\texttt{TokUp}}$, le système atteint une configuration $\gamma \in \Gamma_{\texttt{e}}$ en $O(n)$ rondes. Dans cette configuration, $r$ est le seul nœud activable par la règle $\mathbb{R}_{\texttt{EndBFS}}$, cette règle met $\mathsf{Ecc}_r = e^*(r)$. De plus, $r$ met à jour l'excentricité minimum du réseau (voir variable $\mathsf{NewMinE}_v$), et libère le jeton. Grâce à la propriété 16, $\Gamma_{\texttt{TokDown}} \cup \Gamma_{\texttt{TokUp}}$ est clos, tout nœud $v$ recevant le jeton dans la circulation descendante atteint $\mathsf{Ecc}_v = e^*(v)$. La circulation du jeton prend $O(n)$ rondes, le calcul de $e^*(v)$ prend $O(n)$ pour chaque nœud. Donc, en $O(n^2)$ rondes, le système converge vers $\Gamma_{\texttt{Ecc}}$ et $\Gamma_{\texttt{Ecc}}$ est clos sous $\mathcal{SSCC}$. ■

**Propriété** 20 $\Gamma_{\texttt{Ecc}} \rhd \Gamma_{\texttt{Centers}}$ *en* $O(n^2)$ *rondes, et* $\Gamma_{\texttt{Centers}}$ *est clos sous* $\mathcal{SSCC}$.

**Preuve de la propriété.** La quatrième couche de $\mathcal{SSCC}$ est dédiée à la collecte de l'excentricité minimum du réseau. Cette collecte est prioritaire par rapport au calcul du BFS. Quand un nœud $v$ change son excentricité (variable $\mathsf{Ecc}(v)$), une vague est déclenchée : à partir de $v$ jusqu'à la racine du `Backbone`. Pour cela, l'algorithme utilise la variable $\mathsf{MinEUP}$ (voir propriété 4). Lorsque la racine du `Backbone` reçoit une nouvelle excentricité minimum, elle diffuse le nouveau minimum (voir la règle $\mathbb{R}_{\texttt{MinERoot}}$et la propriété 5) sur le `Backbone`, cette diffusion est assurée de la racine vers les feuilles grâce à la règle $\mathbb{R}_{\texttt{MinEDown}}$ (voir propriété 6). Quand un nœud $v$ execute la règle $\mathbb{R}_{\texttt{MinEDown}}$, si $\mathsf{Ecc}_v = \mathsf{MinE}_v$, alors $v$ sait qu'il est un centre, sinon $v$ n'est pas un centre. Un nœud $v$ qui avait reçu le jeton en descente a sa variable $\mathsf{Ecc}_v = e^*(v)$ avec $e^*(v)$ l'excentricité de $v$ (voir propriété 17), à partir de là à chaque fois qu'il reçoit le jeton en descendant il va maintenir $\mathsf{Ecc}_v = e^*(v)$. Quand l'ensemble des nœuds $v$ du système auront reçu le jeton en descente,

ils auront $\mathsf{Ecc}_v = e^*(v)$, la circulation du jeton en descente et le calcul des BFS maintiendront l'exactitude des excentricités des nœuds, dont les nœuds centres ne changeront pas ($\Gamma_{\texttt{Centers}}$ est clos). Il est à noter que la circulation de jeton (ascendante et descendante) sans calcul de `BFS` prend $O(n)$ ronde. Ici cette circulation est ralenti par le calcul de l'excentricité qui prend à chaque fois $O(n)$ ronde, ce qui nous donne une complexité en $O(n^2)$ rondes. ■

Ceci conclu la preuve du lemme 3.

Pour terminer la preuve du théorème 1, il faut noter que l'ensemble des variables utilisées par notre algorithme utilisent soit un nombre de bits constants soit un nombre logarithmique de bits (taille pour codé les identifiants et les distances). Pour la complexité temporelle nous avons :

Lemme 1 : $\Gamma \triangleright \Gamma_{\texttt{Backbone}}$ en $O(n^2)$ rondes pour $\mathcal{SSCC}$ sous un démon distribué non équitable.

lemme 2 : $\Gamma_{\texttt{Backbone}} \triangleright \Gamma_{\texttt{1-Token}}$ en $O(n)$ rondes pour $\mathcal{SSCC}$ sous un démon distribué non équitable.

lemme 3 : $\Gamma_{\texttt{1-Token}} \triangleright \Gamma_{\texttt{Centers}}$ en $O(n^2)$ rondes pour $\mathcal{SSCC}$ sous un démon distribué non équitable.

Ce qui donne une complexité temporelle de $O(n^2)$ rondes pour la convergence de l'algorithme $\mathcal{SSCC}$ sous un démon distribué non équitable. □

## 3.6 Conclusion

Dans cet chapitre, nous présentons le premier algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum qui tolère tout environnement asynchrone (capturé par un démon distribué non équitable) et utilise un espace mémoire de $O(\log n)$ bits par nœud. Notre algorithme se stabilise au bout de $O(n^2)$ rondes. Cette contribution améliore les résultats existants par un facteur $n$ en ce qui concerne l'espace mémoire.

Ce travail met en évidence les questions ouvertes suivantes. Ces questions se concentrent sur l'optimalité des besoins en mémoire. Les réponses dépendent si nous voulons obtenir un algorithme silencieux auto-stabilisant ou non. Comme notre algorithme est basé sur une circulation de jeton, il n'est pas silencieux. Un travail récent [BT13] présente un algorithme auto-stabilisant non-silencieux d'élection basé sur la construction d'un BFS, cet algorithme utilise $O(\log \log n)$ bits de mémoire par nœud. Ce travail nous conduit à penser que nous pouvons améliorer l'exigence en mémoire de notre algorithme. La première question est donc la suivante : l'algorithme auto-stabilisant non silencieux que nous venons de présenter est-il optimal en mémoire, ou existe-t-il un algorithme non-silencieux pour le problème de l'arbre couvrant de diamètre minimum utilisant moins de $O(\log n)$ bits de mémoire par nœud ?

Cela ouvre naturellement une autre question sur l'optimalité d'un algorithme auto-stabilisant silencieux pour la construction d'un arbre couvrant de diamètre minimum. Une façon attrayante pour répondre à cette question est suggérée dans [BFP14]. En utilisant leur solution, la question est réduite à fournir un réseau de preuves d'étiquetage pour ce problème nécessitant $O(\log n)$ bits de mémoire par nœud. Si un tel réseau d'étiquetage existe, une adaptation pure et simple de notre algorithme auto-stabilisant serait un algorithme auto-stabilisant silencieux et optimal pour la construction d'un arbre couvrant de diamètre minimum.

# Chapitre 4

# Réseaux à faible puissance et à perte

## 4.1 Introduction

Les objets intelligents sont désormais une réalité dans de nombreuses applications réseau, en particulier pour la surveillance d'environnements hostiles où l'intervention humaine est difficile, voir dans certains cas, impossible. Dans ce cadre, les objets intelligents sont déployés sur des zones prédéterminées afin de relever des informations. Ces objets forment un réseau sur lequel les informations collectées sont acheminées par leur intermédiaire jusqu'à une source prédéterminée. Ces objets intelligents sont quelquefois composés de capteurs, ce qui explique que les termes "objets intelligents" ou "réseaux de capteurs" peuvent être utilisés de manière souvent indifférenciée. Ces capteurs peuvent être de différentes sortes comme des capteurs lumineux, des capteurs de mouvements, des capteurs d'humidité, etc. Le faible coût de ces capteurs font qu'ils sont de plus en plus employés afin de réduire le coût de l'intervention ou de la maintenance humaine.

Les caractéristiques de ces réseaux de capteurs sont une faible portée de communication, une puissance de calcul limitée, une faible capacité de mémoire de stockage, et le plus souvent une batterie à durée de vie limitée. Les caractéristiques physiques intrinsèques de ces réseaux conduisent à de forts changements de topologie du réseau et ceci tout au long de la durée de vie de ce réseau. En effet, certains capteurs peuvent disparaître en raison de la durée de vie de la batterie ou de la variation des liens (radio) de communication. Il est important de noter, que ces changements de topologie peuvent entrainer des pertes d'informations. Les réseaux de capteurs sont donc classés dans les réseaux à faible puissance et à perte (en anglais *Low Power and Lossy Networks*, notés *LLN*). Pour ces raisons, les protocoles internet traditionnels doivent être adaptés à de tels dispositifs contraints. Ces protocoles ayant pour principal défi de s'adapter à la variation de la topologie tout en minimisant les messages de contrôle prévu à cette adaptation. En effet, tous les messages de contrôle échangés causent une consommation d'énergie supplémentaire due à des opérations d'interfaces radio-fréquence et de traitement de paquets. Par conséquent, plus faible est le nombre de messages de contrôle, plus grande est la durée de vie globale du réseau.

Le groupe *Internet Engineering Task Force* (IETF) a crée le groupe de travail *ROLL* (Routing Protocol for Low Power and Lossy Networks) dédié à la conception d'un protocole pour les réseaux LLN, le protocole *RPL* [WTB+12a] (Routing Protocol for Low Power and

Lossy Networks). Plus spécifiquement, ce groupe vise à développer un protocole de routage dédié aux réseaux sans fils. Ce protocole de routage doit être multi-sauts, efficace en énergie et évolutif (c'est à dire qu'il doit évoluer avec la taille du réseau). RPL est un protocole IPv6 à vecteur de distance qui construit de manière proactive une topologie logique basée sur un *DODAG* (Destination Oriented Directed Acyclic Graph).

**Organisation du chapitre** Dans ce chapitre, nous définirons dans un premier temps les réseaux de capteurs. Cette définition sera suivie d'une description des couches protocolaires et des technologies utilisées dans de tels réseaux. La section suivante sera quant à elle dédiée au protocole de communication défini par l'IEEE destiné aux réseaux sans fil de la famille des LLN, le protocole 802.15.4. Enfin, la fin du chapitre sera consacrée à une explication détaillée du protocole de routage RPL ainsi qu'à son évaluation des performances.

## 4.2 Réseaux de capteurs

Un réseau de capteurs sans fils est constitué d'un grand nombre de petits dispositifs à bas prix, de faible puissance et équipés d'une batterie dont la durée de vie est la plus part du temps limitée. Ces petits dispositifs appelés capteurs sont essentiellement constitués de capteurs de mesures tels que des capteurs thermiques, capteurs de luminosité, capteurs d'humidité, etc. Ces dispositifs sont destinés à transmettre les données environnementales récoltées par les capteurs. Ils sont également dotés d'émetteurs radio, de microprocesseurs embarqués et d'une batterie. L'émetteur radio est utilisé pour communiquer avec les autres capteurs, ces communications sont établies afin de transmettre les informations collectées à un contrôleur central. L'émission radio de chaque capteur est de faible puissance, ce qui oblige un placement des capteurs à une faible distance les uns des autres pour qu'ils puissent communiquer entre eux. Comme tout réseau de communication, un réseau de capteurs peut être modélisé par un graphe, les capteurs étant les nœuds de ce graphe et les émissions radio les liens de communication. Par la suite, nous utiliserons donc le terme de *nœud* ou *nœud capteur* plutôt que celui de capteur.

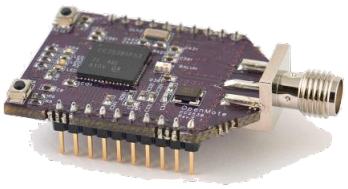


FIGURE 4.1 – Capteur OpenMote-CC2538

### 4.2.1 Domaines d'application

Les réseaux de capteurs sont utilisés dans les applications de surveillance en raison de leur très faible coût et de leur facilité de déploiement dans des environnement hostiles.

**Application environnementale :** La surveillance environnementale est le principal domaine d'application dans les réseaux de capteurs. Dans de telles applications, les nœuds capteurs sont utilisés pour mesurer plusieurs paramètres physiques environnementaux tels que l'humidité et la température dans une forêt, la qualité de l'eau dans les océans, la qualité de l'air dans les villes, etc.

**Application Militaire :** Les capteurs sans fils peuvent être facilement déployés dans des régions en conflits sans aucune infrastructure préalable. Ils peuvent par exemple servir à surveiller les mouvements des adversaires, ils peuvent aussi être utilisés pour protéger des endroits sensibles tels que des zones nucléaires, des gisements de gaz ou de pétrole ainsi que des centres de communications de l'armée.

**Application Médicale :** Les réseaux de capteurs sans fils sont aussi utilisés dans des applications médicales pour surveiller la santé des patients comme la pression sanguine ou la taux de sucre dans le sang. De telles applications permettent de rapidement venir en aide aux patients en cas d'urgence et d'adapter les traitements du patient en fonction des mesures relevées par les capteurs.

**Application industrielle :** Dans des sites industriels tels que les centrales nucléaires, les raffineries ou les sites avec des produits chimiques, les capteurs sont déployés dans des zones inaccessibles (dangereuses en raison d'une forte exposition) par les humains pour surveiller les conditions des équipements industriels et alerter en cas de problème comme par exemple une forte radioactivité ou une fuite de produits chimiques ou de gaz.

**Application domestique :** Les maisons intelligentes utilisent des réseaux de capteurs. Dans un avenir proche, les objects du quotidien seront équipés de capteurs et seront connectés en réseau autonome. Par exemple un capteur sur une machine à café pour surveiller la consommation de café. Un capteur sur la porte d'entrée pour notifier de la présence d'une personne durant son absence ou un capteur dans chaque pièce pour relever la temperature dans la maison.

### 4.2.2 Caractéristiques

Comme les autres réseaux sans fils tels que les réseaux cellulaires et les réseaux maillés, les réseaux de capteurs possèdent leurs propres caractéristiques et contraintes. En voici une liste non exhaustive.

**Batterie :** Les nœuds capteurs sont généralement alimentés par une batterie à durée de vie limitée. Leur remplacement ou leur recharge peut se révéler difficile voir impossible.

**Puissance de transmission :** Les nœuds capteurs sont de petits dispositifs équipés d'une antenne radio émettant à faible puissance.

**Mémoire de stockage :** Les nœuds capteurs disposent d'une faible mémoire de stockage à cause de leur petite taille.

**trafic :** Le trafic majoritaire dans les applications utilisant les réseaux de capteurs est transmit des nœuds capteurs jusqu'à la station de base (souvent appelé *racine*).

**Fiabilité :** Les nœuds capteurs peuvent avoir des pannes dues à des dommages physiques en raison de leur présence dans des environnements hostiles ou à la durée de vie limitée de leur batterie.

**Topologie de routage :** A cause des pannes de nœuds, des problèmes de liens radio, la topologie de routage du réseau change fréquemment.

**Types de données :** En fonction de l'application utilisée, un réseau de capteurs peut prendre en charge plusieurs types de données. Il peut s'agir de données numériques de grandeurs physiques (tels que l'humidité, la chaleur, le rayonnement) ou de données biologiques (tels que la pression artérielle ou le taux de sucre dans le sang). Il peut aussi s'agir de données multimédia (plus volumineuses) lorsque les dispositifs capteurs sont équipés d'une caméra vidéo ou d'un capteur son. Les dispositifs capteurs peuvent aussi être équipés d'accéléromètre pour les applications de détection de mouvement ou dans les réseaux BAN (Réseaux de capteurs corporels).

### 4.2.3 Architecture et couches protocolaires dédiées aux réseaux de capteurs

De même que les réseaux traditionnels, les réseaux de capteurs fonctionnent également selon un empilement de plusieurs couches à partir de la couche physique jusqu'à la couche application. L'empilement de couches pour les réseaux de capteurs diffère de celui du modèle OSI. Il ne comporte que cinq couches principales au lieu de sept, les deux couches restantes étant inutiles et difficiles à implémenter pour un réseau de capteurs. En effet, dans un réseau de capteurs, la couche application fournit uniquement l'affichage et la visualisation des résultats ainsi que des requêtes de contrôle. De plus, implémenter deux couches supplémentaires entrainerait une consommation énergétique supplémentaire.

Les cinq couches protocolaires empilées pour un réseau de capteurs sont : la couche physique, la couche MAC, la couche réseau, la couche transport et la couche application. La Figure 4.2 montre cette superposition de couches. Dans le cas d'une couche réseau IPv6 implémentée au dessus de la couche IEEE 802.15.4, une sixième couche sera ajoutée entre la couche MAC et la couche réseau. Voici une description non exhaustive du rôle de chacune de ces couches dans un réseau de capteurs et les protocoles qui y sont associés.

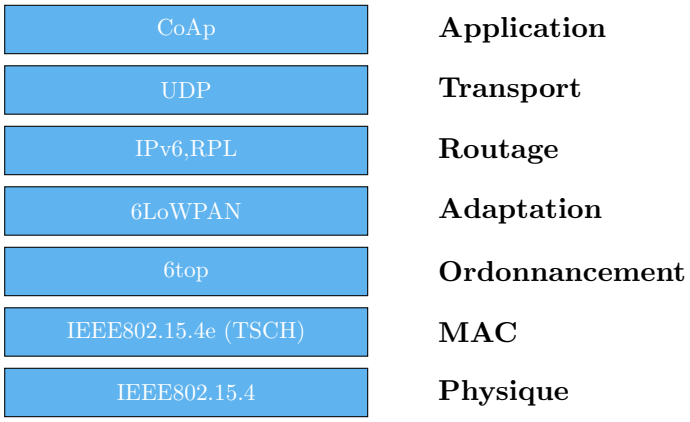



FIGURE 4.2 – Empilement de couches pour un réseau de capteurs 6LoWPAN.

La *couche physique* spécifie les caractéristiques matérielles du réseau. Elle définit aussi la fréquence porteuse et la puissance de transmission, la modulation ainsi que le chiffrement de données. La technologie la plus utilisée dans les réseaux de capteurs pour définir la couche physique est la technologie IEEE 802.15.4 (la section suivante sera consacrée à cette technologie).

La *couche de liaison de données* est responsable de l'accès au lien de communication (le medium de communication) ainsi que de la détection et de la correction d'erreurs

survenues dans la couche physique ou durant la transmission de paquets de données au niveau routage. En outre, cette couche est composée de deux sous couches : la couche de liaison de données (LLC) et la couche de contrôle d'accès au support (MAC). La fonction la plus importante dans la couche de liaison est le contrôle d'accès au support (MAC). Cette couche est généralement appelée dans le cas d'un réseau de capteurs la couche MAC. C'est ainsi que nous la nommerons dans la suite de ce document.

La *couche d'ordonnancement* est implémentée au dessus de IEEE 802.15.4e. Le protocole 6top a été proposé par le groupe 6TiSH pour permettre l'ordonnancement de l'envoie de données. Les nœuds communiquent selon le protocole TDMA (Time Division Multiple Access), ce dernier définit l'ordre de communication entre les nœuds en prenant en compte le trafic réseau.

La *couche d'adaptation* a été conçue pour permettre à des dispositifs s'executant avec la IEEE 802.15.4 de fonctionner avec une couche de routage IPv6. Le groupe 6LoWPAN a défini les mécanismes d'encapsulation et de compression d'entêtes permettant aux paquets IPv6 d'être envoyés ou reçus via le protocole de communication IEEE 802.15.4. Ces mécanismes d'encapsulation et de compression sont définis dans [HT11].

La *couche de routage* sélectionne les chemins que les données doivent empruntées dans le réseau et transmet les données de la source à la destination. Cette couche prend en charge également l'adressage des équipements LLN (notamment les capteurs).

La *couche de transport* fournit une communication transparente entre les capteurs. Elle s'occupe de la fragmentation de paquets et de leur transport. Elle permet le transport de données selon le protocole TCP ou UDP. Le protocole le plus populaire dans les réseaux de capteurs est le protocole UDP car il n'établit aucune connexion, contrairement à TCP, et ne garantit pas la fiabilité des données. En revanche, ce protocole garantit une meilleure consommation énergétique grâce à un faible taux d'émission de messages (car il n'utilise pas de mécanisme de connexion et de contrôle des données).

Enfin, la *couche d'application* est la couche la plus proche des utilisateurs car elle fournit l'interface avec les applications. Elle permet ainsi l'implementation de différentes options telles que la diffusion de requêtes, la localisation des nœuds, la synchronisation des nœuds et la sécurité du réseau. Cette couche prend également en charge le protocole de transfert web COAP.

## 4.3 La technologie 802.15.4

IEEE 802.15.4 est un protocole de communication proposé pour des applications qui requièrent une faible bande passante, une faible capacité de calcul et une puissance d'émission limitée. Elle répond par conséquent à une partie des exigences des réseaux LLN. Dans la suite, nous décrivons les types de dispositifs 802.15.4, les fonctions dans la couche physique et MAC et les modes de communications utilisés dans la IEEE 802.15.4.

### 4.3.1 Dispositifs 802.15.4

La norme définit au niveau de la couche MAC trois types de dispositifs qui utilisent tous le même canal physique pour communiquer :

Le premier dispositif est le **coordinateur** qui gère les fonctions de haut niveau du réseau comme l'authentification et la sécurité. Il existe un seul coordinateur pour tout le

réseau. Il s'agit dans la plupart du temps du nœud puits (appelé*la racine*).

Les dispositifs à fonction complète (Full Function Devices), appelés **FFD**, sont des dispositifs dotés de toutes les fonctions. Ils peuvent par conséquent jouer le rôle d'un coordinateur ou d'un dispositif relié à un capteur. Ils aident des dispositifs plus restreints en fournissant des fonctions telles que la coordination du réseau, l'acheminement de paquets ou l'interaction avec d'autres types de réseaux. Les FFD sont les dispositifs les plus utilisés dans les réseaux de capteurs.

Les dispositifs à fonction réduite (Reduce Function Device), appelés **RFD**, possèdent des fonctions limitées, contrairement aux FFD. Les RFD ne peuvent être que des nœuds feuilles (en anglais *E*nd Devices) dans l'arborescence construite car les nœuds **RFD** ne communiquent pas entre eux, ils ne peuvent communiquer qu'avec des FFD afin d'acheminer leur trafic.

### 4.3.2 Spécifications de la IEEE 802.15.4

Le standard IEEE 802.15.4 spécifie les règles et normes de communication pour les couches physique et MAC [MKHC07]. La **couche physique** de la IEEE 802.15.4 contient l'émetteur récepteur radio avec un mécanisme de contrôle de la qualité du signal et de détection d'énergie. Elle émet, selon deux types de fréquences. L'émission à basses fréquences pour une plus grande portée et une faible perte de propagation. L'émission à hautes fréquences pour une faible latence et des cycles de transmission/reception plus courts. La **couche MAC** de la IEEE 802.15.4 quand à elle gère les mécanismes d'accès au canal (CSMA/CA), les balises de contrôle et la validation de trames. Cette couche fonctionne en deux modes de communication : le *beacon enabled mode* avec envoie de balises et le *non beacon enabled mode* sans envoie de balises. La sous-section suivante décrit ces deux modes.

### 4.3.3 Modes de communication de la IEEE 802.15.4

Dans un réseau IEEE 802.15.4, tous les capteurs fonctionnent de façon distribuée. Par conséquent, ils sont indépendants les uns des autres. Pour pouvoir communiquer sur le réseau, ils doivent connaitre à quel moment se réveiller pour transmettre. A cet effet, le protocole IEEE 802.15.4 propose deux de modes de communication, un avec envoi de balises de contrôle et un autre sans envoi de balises de contrôle. Dans le mode de communication avec envoi de balises, appelé *beacon enabled mode*, les nœuds envoient périodiquement des messages (balises) afin de se synchroniser avec les autres nœuds. Avec l'envoie de balises, tous les dispositifs sont informés de la durée d'activité de leur voisins et savent donc à quel moment ils peuvent transmettre des données. Le *Beacon enabled mode* fonctionne sans CSMA/CA. Ce mode de communication a l'avantage de minimiser grandement la collision de données.

Dans le mode de communication sans envoi de balises, appelé *non beacon enabled mode*, les dispositifs ne sont pas synchronisés entre eux car ils n'émettent pas de balises. Cependant, les dispositifs transmettent les données en utilisant le mécanisme *CSMA/CA*. Ce dernier se base sur l'écoute du canal : lorsqu'un nœud veut transmettre une donnée, il vérifie d'abord si le canal est libre. Si c'est le cas, il transmet, sinon il attend une période définie par le protocole IEEE 802.15.4. Dans le mode sans envoi de balises, les nœuds dorment la majorité du temps. Il se réveillent périodiquement pour écouter le canal

et transmettre des données : *duty cycle*. Le *non beacon enabled mode* permet une faible consommation énergétique puisque les nœuds n'envoient pas de balises. De plus, avec ce mode, le canal de communication n'est utilisé que pour transmettre des données.

## 4.4 Protocole RPL

Le groupe de travail IETF ROLL a été chargé de définir un protocole qui prend en compte les exigences spécifiques des réseaux LLN. Ce groupe a développé le protocole de routage **RPL** : protocole IPv6 de routage pour des réseaux à faible puissance et à perte. Le protocole RPL est implémenté au niveau routage (route over), il est donc défini au niveau de la couche réseau dans l'architecture IP.

Cette architecture permet à chaque nœud $v$ de connaitre les nœuds qui sont accessibles en une seule transmission, ces nœuds forment l'ensemble des voisins de $v$. RPL est un protocole de routage IPv6 à vecteur de distance qui construit une topologie logique appelée **DODAG** : Destination Oriented Directed Acyclique Graph. En construisant ce DODAG, RPL permet de transporter des paquets IPv6 à leur destination à travers une ou plusieurs routes. Les spécifications de ce protocole sont définis dans [WTB+12b].

### 4.4.1 Trafic supporté

L'un des principaux avantages du protocole RPL est qu'il supporte plusieurs types de trafics réseau : le trafic multi-points à point, le trafic point à multi-points et le trafic point à point. Le protocole RPL répond ainsi aux exigences du transfert d'informations dans les réseaux de capteurs d'aujourd'hui.

Le trafic dominant dans les réseaux de capteurs est le *trafic multi-points à point* (noté MP2P). Ce trafic transite à partir de capteurs vers la racine. Il est appelé le trafic ascendant et est acheminé à travers le DODAG construit.

Le trafic envoyé de la racine du DODAG vers un sous ensemble de capteurs du réseau est appelé le *trafic point à multi-points* (noté P2MP). RPL utilise un mécanisme d'annonce de destination qui utilise des messages de contrôle appelés DAO pour fournir des routes ascendantes vers des nœuds feuilles. Les destinataires du trafic MP2P sont des nœuds qui doivent par exemple fournir une connectivité pour internet ou à des réseaux à infrastructure privée.

Le trafic acheminant les données d'un nœud vers un autre nœud dans le réseau s'appelle le *trafic point à point* (noté P2P). Avec ce type de trafic, un nœud $v$ envoie un paquet vers un autre nœud $u$, ce paquet transite jusqu'au plus proche ancêtre commun de $v$ et $u$ et à partir de cet ancêtre, il est redirigé vers le nœud $u$.

### 4.4.2 La fonction objective

La fonction objective indique la méthode qui doit être utilisée pour construire un DODAG. Plus précisément, elle permet pour chaque nœud de sélectionner, entre autres : le DODAG à joindre, le nombre de parents qu'il peut prendre, son parent préféré, ses successeurs possibles et de calculer son rang dans le DODAG. La fonction objective définit comment les métriques et contraintes de routage doivent être utilisées pour calculer le rang d'un nœud et sélectionner en fonction de ce rang le nœud parent. Le *rang* d'un nœud est

sa distance relative à la racine du DODAG, il est utilisé pour établir la position relative du nœud par rapport aux autres nœuds. La fonction objective, le rang ainsi que les métriques et les contraintes de routage sont annoncées dans les messages DIO. Seulement deux fonctions objectives ont été définies jusqu'à présent pour le protocole RPL : la fonction objective zéro [Thu12] et la fonction objective MRHOF [GL12]. La fonction objective zéro (notée *OF0* pour Objective Function Zero) est la fonction objective par défaut définit dans RPL. Avec la *OF0*, un nœud calcule son rang en ajoutant une valeur au rang de son parent. La valeur ajoutée au rang de son parent représente la métrique de routage du parent ou du lien vers le parent (la métrique est égale à 1 si elle représente le nombre de sauts jusqu'à la racine). La fonction objective d'hystérisis minimum (notée *MRHOF* pour Minimum Hysterisis Objective Function) minimise quand à elle la métrique de routage. Avec la *MRHOF*, un nœud choisit son parent avec pour but la minimisation de la métrique vers la racine du DODAG.

### 4.4.3 Métriques et contraintes de routage

Les nœuds RPL supportent un ensemble de métriques et de contraintes de routage annoncées dans les messages DIO. Une métrique ou contrainte de routage est spécifique pour chaque nœud ou lien de communication dans le DODAG. Une métrique de routage est une quantité scalaire utilisée pour choisir au mieux un parent. Au contraire, une contrainte supprime de la selection un parent ou un lien par exemple pour non respect d'un ensemble de contraintes.

Le rang d'un nœud peut représenter une contrainte de routage ou une métrique de routage (potentiellement accumulative). RPL spécifie deux types de métriques (ou contraintes), une sur le nœud et la deuxième sur l'état du lien. Une métrique de routage peut représenter par exemple l'état du lien radio d'un nœud vers son parent, l'énergie résiduelle d'un nœud, le nombre de sauts jusqu'au parent ou la latence (délai accumulé) du lien de communication. Dans la suite, nous faisons une brève description de la majorité de ces métriques. En revanche, nous allons faire une description détaillée de la métrique ETX car cette métrique est la plus utilisée dans les implémentations du protocole RPL. Pour plus de détails sur ces métriques, le lecteur peut se référer à [VKP+12].

La métrique *état du nœud* fournit des informations sur les caractéristiques d'un nœud telles que la condition du CPU ou de la mémoire.

La métrique *énergie du nœud* désigne les ressources en énergie du nœud. Par exemple, l'énergie résiduelle d'un nœud.

La métrique *Nombre de sauts jusqu'au nœud* calcul le nombre de nœuds traversés (sauts) sur la route entre la source et la destination.

La métrique *débit du lien* communique la gamme de débit que le lien peut manipuler en plus du débit actuel.

La métrique *latence du lien* indique la latence du chemin, c'est à dire, le temps nécessaire à un paquet pour atteindre la destination à travers le chemin.

La métrique *couleur du lien* est utilisée pour donner des couleurs aux liens et désigner par conséquent les liens à emprunter ou à éviter.

La métrique *niveau de qualité du lien* quantifie la fiabilité du lien en utilisant une valeur discrète entre 0 et 7.

La métrique *E*TX (pour Expected Transmission Count) est la plus couramment utilisée

dans le protocole RPL. Cette métrique traduit la qualité du lien radio en se basant sur les informations que lui transmet la couche MAC. Ces informations représentent la perte de paquets entre un nœud et son voisin. La perte de paquets d'un nœud vers son voisin est notifiée dans la couche MAC grâce à mécanisme d'acquittement. Le nombre de paquets perdus est enregistré et la métrique ETX est calculée à partir de ce nombre. Ainsi, plus le nombre de paquets perdus augmente, plus la métrique ETX se dégrade. Dans le cas où la métrique ETX s'améliore (quand il y a moins de perte de paquets), le nœud peut décider de changer de parent (dont la métrique s'est amélioré) ou de conserver le parent auquel il est affilié (si la métrique vers ce parent reste meilleure que les autres). Pour une description détaillée de la métrique ETX, voir [CABM03].

### 4.4.4 Les messages de contrôle

Pour la construction et la maintenance des routes du DODAG et afin d'acheminer les différents types de trafics, RPL utilise des messages de contrôle ICMPv6. Ces messages de contrôle utilisent l'entête fournit par les messages ICMPv6 en plus d'autres informations nécessaires pour la construction du DODAG telles que le rang et la fonction objective.

Un message ICMPv6 envoyé dans le protocole RPL est encapsulé lui même au niveau de la couche physique dans une entête physique de 127 octets, d'une entête MAC, éventuellement d'une entête de sécurisation de la couche liaison données de 21 octets, d'une entête ICMPv6 de 40 octets au niveau de la couche routage. Le reste du message ICMPv6 sont les données utiles (ou les options). La Figure 4.3 représente une trame d'un paquet de contrôle ICMPv6.

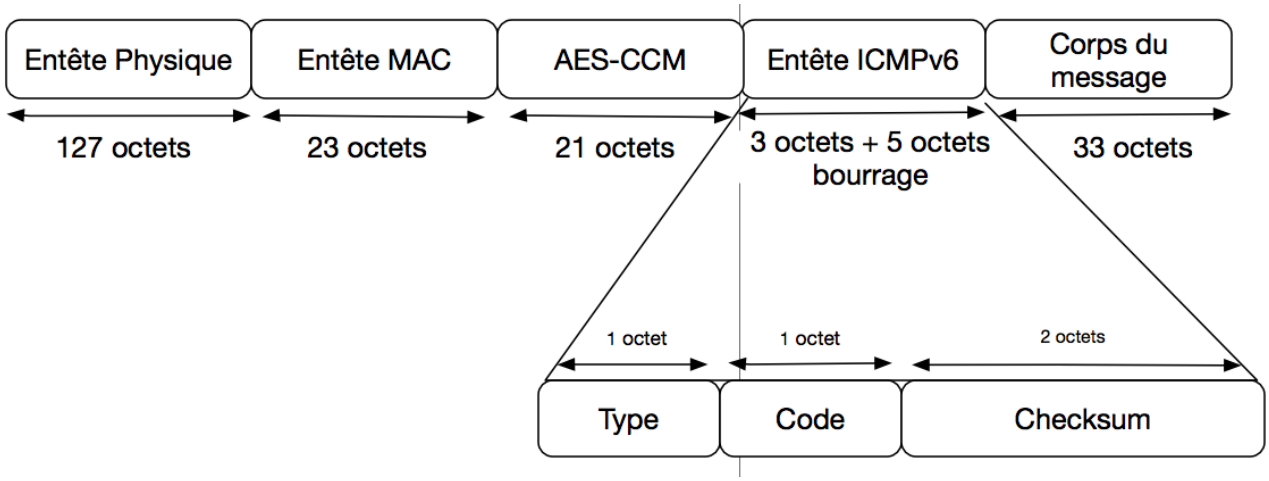



FIGURE 4.3 – Trame d'un paquet de contrôle ICMPv6

Un message de contrôle RPL est constitué d'une entête IMPCv6 suivie par un corps (*base* et *options*) de message (voir Figure 4.4). Le protocole RPL définit principalement quatre types de messages de contrôle ICMPv6 : DIS, DIO, DAO, DAO-ACK. Ce type de message est spécifié dans l'entête du message ICMPv6 dans un champ spécifique. La Figure 4.4 représente un message ICMPv6 type avec l'emplacement du type de message RPL.

Dans l'entête ICMPv6 d'un message de contrôle RPL, le champ *type* (sur 8 bits) spécifie

le type de message ICMPv6. Dans le cas d'un message de contrôle RPL, c'est le type 155. Le champ *code* sur (8 bits) identifie le type de message de contrôle RPL. C'est à dire : un message DIO, DAO ou tout autre type de messages de contrôle RPL. Le champ *Checksum* sur 16 bits est réservé à des options.

Le corps d'un message de contrôle RPL est constitué d'une base de message et d'un ensemble d'options. Dans le cas d'un message de contrôle RPL sécurisé est rajouté un champ sécurité.

Les informations (différentes pour chaque type) relatives à chaque message de contrôle RPL sont spécifiées dans la base du message. Ces informations sont, entre autres, l'identifiant de l'instance RPL, le rang du nœud qui a envoyé le message, la fonction objective et l'adresse IPv6 de la racine du DODAG.

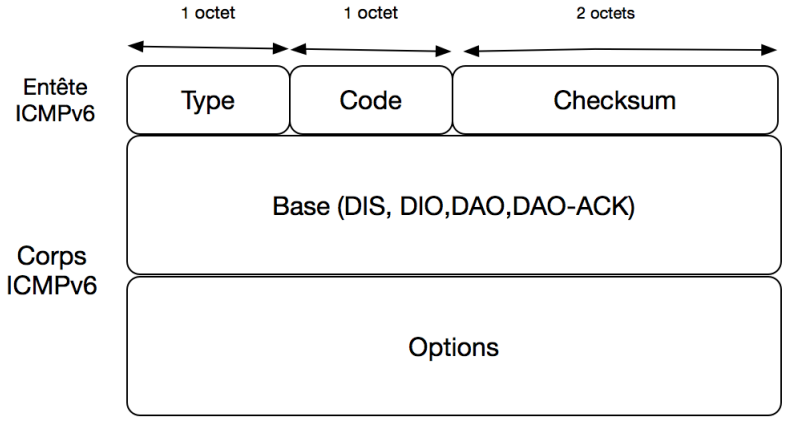



FIGURE 4.4 – Message ICMPv6

Pour plus de détails sur le contenu des entêtes ICMPv6 des messages de contrôle RPL, voir [WTB+12b]. Nous allons à présent décrire plus en détails le rôle de ces messages dans le protocole RPL.

— Les messages *DIS* (pour DODDAG Information Solicitation) sont utilisés pour demander à un nœud RPL des informations sur un DODAG. Par exemple, un nœud peut envoyer des DIS pour déterminer les DODAG les plus proches (et éventuellement décider de s'y associer) ou pour solliciter un message DIO.

— Les messages *DIO* (pour DODAG Information Object) quant à eux, servent à construire et à maintenir des DODAG. Ils permettent à un nœud de découvrir une instance RPL, de comprendre ses paramètres de configuration, de sélectionner un parent dans le DODAG et de maintenir ce DODAG.

— Les messages *DAO* (pour Destination Advertisement Object) sont envoyés pour acheminer des informations de destination des parents vers les enfants sur le DODAG construit au préalable.

— Les messages *DAO-ACK* (pour Destination Advertisement Object acknowledgment) sont une réponse en forme d'un paquet unicast envoyé par un parent pour acquitter le message DAO de son enfant.

Un message de données envoyé dans le protocole RPL est encapsulé au niveau de la couche physique dans une entête physique de 127 octets, d'une entête MAC, éventuellement d'une entête de sécurisation de la couche liaison données de 21 octets, d'une entête IPv6

et d'une entête RPL au niveau de la couche routage et d'une entête UDP de 8 octets (ou TCP de 20 octets) au niveau de la couche transport. Le reste du message ICMPv6 sont les données utiles (ou les options) de 21 (ou de 33 octets).

### 4.4.5 DODAG

Le protocole RPL spécifie que plusieurs DODAG peuvent être présents en même temps. Cela permet de transporter différents types de trafics tels que des trafics qui minimisent la latence, assurent la fiabilité de l'information ou réduisent la consommation énergétique.

Cette topologie logique est un arbre couvrant où la racine est le nœud capteur appelée *DODAG root*. Chaque nœud choisit parmi ses voisins un ensemble de successeurs (*F*easible Successors dans RPL) et un parent (parent préféré dans RPL). L'ensemble des successeurs étant les voisins proposants une "bonne" métrique, et le parent préféré étant le voisin optimisant cette même métrique (voir Figure 4.5).

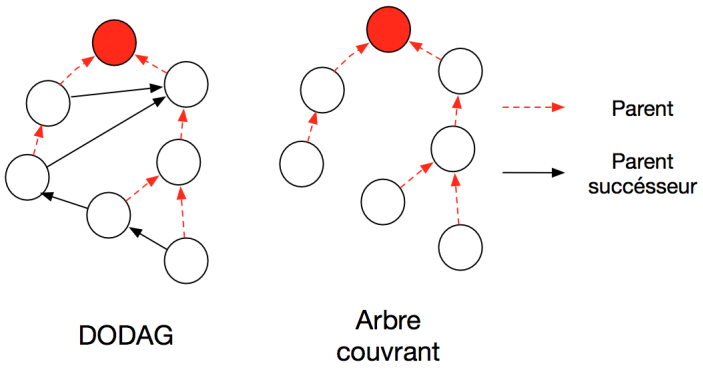



FIGURE 4.5 – DODAG comparé à un arbre couvrant

#### Construction d'un DODAG

Pour construire un DODAG, la racine envoie un message DIO contenant des informations comme l'identifiant du DODAG et la métrique utilisée à tous ses voisins. Lorsqu'un nœud reçoit un message DIO d'une racine, il prend la décision basée sur certaines règles (politique locale) pour savoir s'il doit rejoindre le DODAG ou non. Dans le cas ou le nœud rejoint le DODAG, il détermine tout d'abord ses voisins. Il choisit ensuite parmi ses voisins son parent en fonction de la métrique de routage et de la fonction objective (annoncées dans le DODAG) et calcule son rang dans le DODAG. Le nœud choisit aussi parmi son ensemble de voisins des parents de substitution (appelé *F*easible Successors). Dans le cas, où le parent n'est plus joignable, un des parents de substitution le remplacera dans le DODAG. Le nœud transmet ensuite à son tour un message DIO à ses voisins. Ce processus est répété par l'ensemble des nœuds du graphe, jusqu'à ce que le DODAG soit construit.

Le DODAG est ainsi construit, de saut en saut, de la racine jusqu'aux feuilles (Le pseudo code de l'algorithme 5 résume ce processus). Il existe une route entre chaque nœud du DODAG et la racine. Un nœud peut ainsi envoyer des paquets de données à la racine

en envoyant le paquet à son parent. Les paquets sont transmis par conséquent saut par saut jusqu'au la racine du DODAG.

**Algorithm 5:** Construction du DODAG

**Initialization :**
  **si** $v = racine$ **alors**
    **Envoie** DIO aux voisins de $v$

**Reception** d'un DIO envoyé par le nœud $u$ :
  $N(v) := \{u\} \cup N(v)$
  **si** $rang(u) < rang(p_v)$ **alors**
    $parent_v := u$
    **Mise à jour** de $rang(v)$
  **sinon**
    **si** $|sucesseur(v)| < B$ **alors**
      $successeur(v) := sucesseur(v) \cup \{u\}$
    **sinon**
      **si** $\exists w \in sucesseur(v) \mid rang(w) > rang(u)$ **alors**
        $successeur(v) := sucesseur(v) \cup \{u\} \setminus \{w\}$
  **Envoie** DAO à $u$
  **Envoie** DIO à $N(v)$

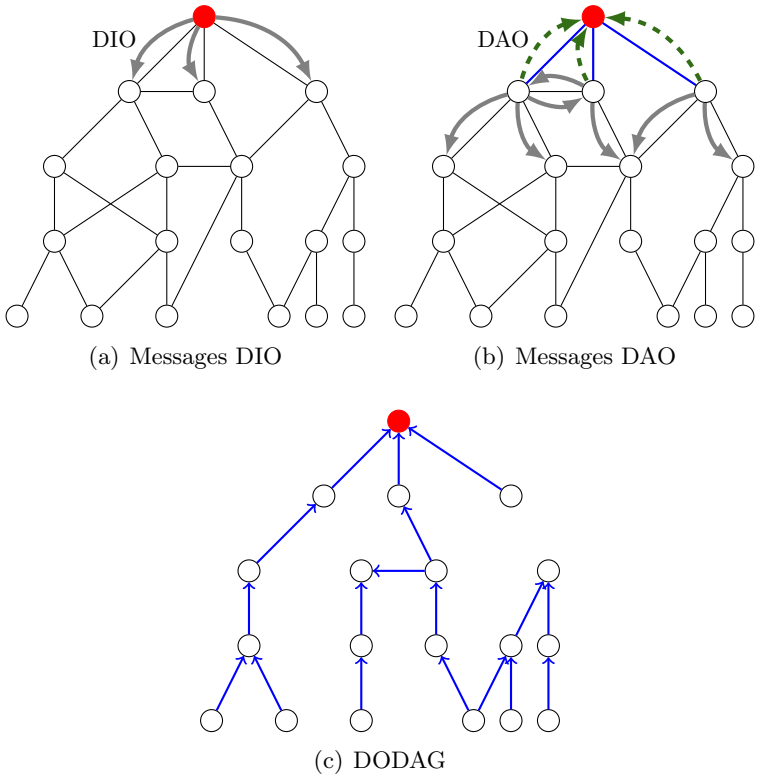
**Reception** d'un DAO envoyé par le nœud $u$ :
  $Enfants(v) := Enfants(v) \cup \{u\}$
  **Envoie** DAO-ACK à $u$
  **Mise à jour** de l'information de destination
  **Envoie** DAO à $parent_v$

La Figure 4.6 présente la construction d'un DODAG en utilisant le métrique qualité du lien (ETX). La racine initie la construction du DODAG en diffusant des messages DIO 4.6(a). Les voisins de la racine choisissent leur parent, envoient le DAO à leur parent et envoient à leur tour leur propre DIO 4.6(b). Le processus continue jusqu'à la construction définitive du DODAG 4.6(c).

### Les routes descendantes

Il est parfois nécessaire d'acheminer d'autres type de trafics tels que le trafic point à point et le trafic multi-points à point. Ce trafic peut provenir de l'extérieur du réseau LLN depuis une racine ou un nœud intermédiaire vers les feuilles. La construction du DODAG ne spécifie que les chemins (routes construites par le DODAG) pour les trafics multi-points à point (*convergecast*). Ces chemins représentent les routes ascendantes. Pour faire transiter les autres types de trafics, RPL doit donc définir des routes descendantes qui seront construite sur le DODAG initial. A cet effet, RPL utilise les messages DAO et DAO-ACK pour la construction de ces routes. Pour la construction des routes descendantes dans RPL, deux modes d'opération (*MOP* pour Mode Of Operation) sont définis.

Le mode avec capacité de stockage des tables de routage appelé *Storing mode* permet au nœud de stocker les adresses IPv6 de leurs enfants dans des tables de routages. Un nœud envoie un message DAO unicast à son parent, ce message contient une information de préfixe. Le nœud qui reçoit le message DAO transfert l'information et ajoute une entrée




(a) Messages DIO (b) Messages DAO

(c) DODAG

FIGURE 4.6 – Construction d'un DODAG selon la métrique ETX

dans la table de routage. Le processus continue jusqu'à ce que l'information arrive à la racine et qu'un chemin complet de préfixe soit établi. Le message est ensuite envoyé le long de la route descendante. Le mode storing requiert plus de mémoire pour enregistrer les tables de routage.

Dans le second mode, appelé *non-storing mode*, seule la racine a la capacité de stocker les informations de routage, le reste des nœuds sont en mode non storing. Puisque les nœuds (excepté la racine) n'enregistrent pas les routes descendantes, un paquet envoyé remonte tout le chemin jusqu'à la racine. Ensuite, la racine construit le chemin jusqu'à la destination et envoie le paquet à la destination sur le chemin construit. Dans ce mode, un message DAO est unicast et possède toujours l'adresse de la racine. Le nœud intermédiaire qui reçoit un message DAO ajoute son propre DAO et transmet le DAO composé à son parent, qui à son tour le transmet à son parent, ainsi de suite, jusqu'à arrivé à la racine. La racine utilise ensuite l'information dans le DAO composé (destinée aux parents de chaque nœud dans la route) pour reconstituer une route descendante à un nœud en utilisant les DAO. Le mode non storing requiert plus de puissance et de bande passante pour transporter les paquets de plus en plus grands. car les nœuds intermédiaires n'ont pas la capacité de stocker les routes.

### Maintenance du DODAG et évitement de cycles

Après la construction du DODAG, le changement de topologie est peut être fréquent avec RPL (à cause de la variation de la connectivité dans les liens radio). Pour éviter que

des cycles apparaissent, RPL définit un mécanisme d'évitement de cycles, de détection de cycles et de reconstruction basée sur la valeur du rang. Le mécanisme d'évitement et de détection de cycles est le suivant. Tout d'abord, le mécanisme spécifie que le rang des nœuds doit strictement augmenter par rapport à leurs parents. Ensuite, lors du routage des données, un nœud peut détecter une inconsistance entre le rang du nœud qui envoie et la direction du paquet envoyé, c'est à dire ascendante ou descendante. Par exemple, lorsqu'un nœud reçoit un paquet, marqué comme trafic ascendant, d'un voisin de rang inférieur au sien, il conclut qu'il y a un cycle dans le DODAG. Enfin, lorsqu'un cycle est détecté, le nœud lance la reconstruction des routes concernées (Voir [WTB+12a] pour la description du *Local Repair* dans RPL).

### 4.4.6 Le trickle timer

Le *trickle timer* [LCH+11] est un mécanisme qui gouverne l'envoie de messages DIO afin de minimiser les messages de contrôle et de préserver l'énergie des nœuds. Il contrôle à quel moment envoyer des messages DIO. Autrement dit, quand la topologie est stable, le trickle timer autorise peu d'envoi de messages DIO mais quand il y a une inconsistance telle qu'un changement de topologie ou un cycle, le trickle timer autorise l'envoie massif de messages DIO. Le trickle timer fonctionne en initialisant une valeur qui augmente jusqu'à une valeur maximale donnée. Cette valeur sera ré-initialisée dans le cas d'une inconsistance pour que de nouveaux messages DIO soient envoyés.

## 4.5 Evaluation des performances du protocole RPL

Depuis la standardisation du protocole RPL, celui-ci a été largement évalué par la communauté. Par simulation [TdOV10b, RSA12a, ITN13] ou par experimentation [GK12, CHP11]. Le but de ces études était de comprendre le mécanisme du protocole RPL, ses atouts et ses faiblesses.

Les auteurs de [CHP11] étudient l'applicabilité et les limites du protocole RPL. Ils implémentent RPL dans Contiki et l'évaluent sur une plateforme réelle. Ce papier fournit une évaluation complète sur les faiblesses et les limites de RPL, notamment la surcharge en messages DAO et l'implémentation complexe de RPL.

Le protocole RPL a aussi été évalué dans un environnement mobile [LSD+12]. Les auteurs du papier évaluent RPL dans des réseaux véhiculaires (VANET : Vehicular Ad hoc Networks). Les simulations ont été faites en désactivant le trickle timer, les auteurs ont ensuite implémenté un nouveau mécanisme afin de détecter et d'éviter les cycles. Ils ont conclu que le trickle timer ne convient pas à un environnement hautement mobile car il ne permet pas à la topologie de s'adapter rapidement aux changements dus à la mobilité.

Les auteurs de [TdOV10a] ont évalué le protocole RPL dans une simulation basée sur un scénario de déploiement dans la vie réelle. Ils ont montré que le rétablissement des routes s'effectue dans 90% des cas de perte de liens. Ils ont aussi montré que le trickle timer aide grandement dans la réduction du nombre de messages de contrôle échangés entre les nœuds.

Une étude sur le processus de réparation des routes RPL a été étudié dans [KSS12]. Les auteurs ont implémenté un nouveau mécanisme sur ContikiRPL afin d'étudier le comportement du mécanisme de réparation locale (*local repair*) de RPL et la capacité du réseau

à reconstruire les routes.

Dans [GK12], les auteurs présentent une étude sur le protocole RPL. Ils évaluent les performances du protocole en termes de taux de perte de paquets, d'énergie et de délai. Les auteurs concluent que le taux de perte de paquets du protocole est due à la forte instabilité de la topologie. Instabilité aussi mis en avant dans [ITN13]. En conséquence, dans [GK12], il est proposé de concevoir une nouvelle métrique de routage pour construire des routes plus adaptées à l'instabilité des liens radio.

### 4.5.1 Autres protocoles de routage

Dès l'apparition des réseaux de capteurs, il était évident que les protocoles de routages traditionnels n'étaient pas adaptés puisqu'ils ne prenaient pas en compte les contraintes énergétiques et de stockage. Par conséquent, la communauté a fait de très grands efforts pour proposer des protocoles qui prennent en compte ces contraintes. Il existe un grand nombre de protocoles de routage qui ont été proposés dans la littérature comme LEACH, AODV, LOADng et CTP. Le protocole RPL a été évalué et comparé à d'autres protocoles de routage dédiés aux réseaux LLN [HC11, VTD14, LCC$^+$12]. La plupart de ces études ont montré que RPL est un protocole prometteur avec la meilleure performance en termes de délai, d'énergie consommée et de surcharge en message de contrôle. Nous allons décrire rapidement ces autres protocoles de routage et faire leurs comparaison avec le protocole RPL.

*LEACH* est un protocole de routage hiérarchique organisé en cluster dédié aux réseaux de capteurs. Dans ce protocole, les nœuds transmettent les données aux têtes des clusters qui à leur tour transmettent les données à la racine. LEACH a l'avantage de proposer un mécanisme TDMA pour organiser la communication des nœuds avec le cluster et un mécanisme CDMA pour éviter les interférences entre les clusters. LEACH a le désavantage de consommer beaucoup d'énergie car les nœuds transmettent la plupart du temps avec une grande puissance.

*LOAD/LOADng* ou sa plus récente version The Lightweight On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng) [CdVJY$^+$12], est un protocole de routage réactif qui construit les routes vers une destination donnée uniquement à la demande afin de minimiser la surcharge en messages de contrôle. Le protocole LOADng est une adaptation du protocole Ad hoc On-Demand Distance Vector Protocol (AODV) [PBRD03] pour le rendre compatible avec les contraintes des réseaux LLN. L'un des principaux désavantage de LOADng est le délai nécessaire à la construction des routes. Ce délai entraine un retard dans la délivrance de paquets. Un autre inconvénient de LOADng est l'inondation du réseau par les messages de contrôle et par conséquent la collision des messages et l'augmentation de la consommation énergétique.

Dans [HC11] les auteurs ont comparé RPL et LOAD dans le cas d'un trafic bidirectionnel (envoie et acquittement de la donnée) en utilisant le simulateur NS2. Les résultats de simulations ont montré que les deux protocoles ont un taux de délivrance de paquets raisonnable. Mais que LOAD produit un faible taux de messages de contrôle par rapport à RPL (due aux messages DAO) tout en étant moins contraignant dans le type de trafic et les topologies supportées. LOAD ne fournit pas forcément de routes optimisées mais est le plus adapté au trafic bidirectionnel. Alors que RPL fournit les meilleures performances dans les cas d'un trafic convergent vers un dispositif central. En revanche, les auteurs dans [VTD14] ont démontré qu'avec RPL, la surcharge en messages de contrôle depend

fortement de son implémentation. Un choix judicieux des paramètres de simulation peut considérablement réduire la surcharge en messages de contrôle. Ils ont donc comparé le protocole RPL et LOADng dans le cas d'une automatisation domestique (home automation). Les auteurs ont montré que RPL fournit des routes plus courtes et un arbre couvrant moins profond que LOADng. Ils ont aussi montré que le délai avec RPL est plus court. Par ailleurs, RPL requiert une certaine complexité d'implémentation même s'il utilise moins de mémoire que LOADng.

Le protocole *CTP : Collection Tree Protocol* [GFJ+13] est un protocole de routage dédié aux réseaux de capteurs qui construit un arbre couvrant à partir de chaque nœud du réseau jusqu'à la racine. Comme RPL, le protocole CTP utilise la métrique ETX pour la construction des routes. Il évite aussi les cycles en utilisant un mécanisme qui se base sur la valeur de la métrique ETX. Le protocole RPL est grandement inspiré de ce protocole, il reprend les bases du protocole CTP mais RPL a l'avantage de posséder plus de fonctionnalités telles que le support du trafic point à multi-points et le trafic point à point. RPL et CTP ont été comparés en considérant la taille du réseau et le volume du trafic de données [LCC+12]. Dans ce papier, les auteurs ont comparé RPL et CTP avec le système d'exploitation Contiki et le simulateur Cooja. Ils ont montré que dans le cas d'un réseau de petite taille, CTP offre de meilleures performances que RPL en termes de taux de réception de paquets et de consommation énergétique. En revanche, dans un réseau plus large, RPL possède un meilleur taux de réception de paquets ainsi qu'une plus faible consommation énergétique. RPL conserve également ces propriétés quand le réseau passe à l'échelle (autrement dit, dans un réseau plus grand avec trafic de données croissant) alors que CTP se dégrade fortement.

Les performances des protocoles de routage AODV, CTP et DSR ont été comparées à RPL dans [RSA12b]. Les résultats de simulations ont montré que RPL était le meilleur protocole de routage en termes de latence, de surcharge en messages de contrôle et de consommation énergétique. En revanche, les résultats ont montré que RPL générait une perte de paquets conséquente par rapport aux autres protocoles.

## 4.6 Améliorations du protocole RPL

Depuis la standardisation du protocole RPL, plusieurs travaux ont été proposés pour améliorer ses performances, la plupart de ces travaux ont été consacrés à l'amélioration et à la proposition de métriques de routage [GKBA14, MFAM13]. Par contre, peu de solutions ont abordé la construction de l'arbre couvrant par le protocole RPL [LHD13]. On peut citer aussi, des solutions inter-couches [VRG+14, GMFL14]. Enfin, plusieurs travaux ont été consacrés au trickle timer et à la mobilité [VRG+14, ST11, CPMD15]. Nous allons décrire dans la suite quelques unes de ces solutions.

Dans [GKBA14], les auteurs ont donné une nouvelle fonction objective qui combine plusieurs métriques telles que la ETX, le délai de bout en bout, le nombre de sauts et le niveau de batterie des nœuds afin de calculer la meilleure route. Dans [MFAM13] sont proposées deux nouvelles métriques de routage qui s'adaptent au comportement dynamique de la couche MAC et routage dans les réseaux LLN. La première garantit la fiabilité du chemin. La deuxième minimise l'énergie consommée dans le réseau.

Les auteurs dans [PTD11] ont proposé un schéma de routage opportuniste. Un nœud choisit dynamiquement le prochain saut en estimant la qualité du lien et le meilleur parent juste avant la transmission. Le trafic est plus uniforme car les données sont transmises à

un parent différent à chaque fois au lieu d'un seul parent. Une extension de RPL, appelée P2P-RPL [EMM11], a pour but de construire de façon réactive des routes point à point. Les auteurs ont évalué RPL et P2P-RPL sur la plateforme Iot-lab. Les expérimentations ont montré que P2P-RPL établi le plus court chemin et réduit le trafic jusqu'à la racine. Les auteurs de [WTZ+10] ont modifié le protocole RPL afin de l'adapter dans une grille intelligente AMI (Advanced Metering Infrastructure). Ils ont proposé un nouveau mécanisme pour construire les routes descendantes tout en réduisant la surcharge en messages de contrôle. Les résultats de simulations ont montré que l'adaptation de RPL avait un taux de délivrance de paquets ainsi qu'un délai préformant non sensibles à la distance.

Dans [GMFL14], les auteurs ont considéré un protocole MAC asynchrone avec des cycles d'endormissement et de réveil (duty cycle). Ils ont proposé une solution inter-couches où le DODAG construit par le protocole RPL est exploité pour aligner la période de réveil des enfants avec leur parent afin de réduire le délai. L'article [WAA11] est consacré à un protocole MAC à préambule échantillonné pour améliorer les performances du protocole RPL. Le schéma proposé est adapté aux liens à perte puisque la selection du parent est dynamique et est basée sur les conditions du canal et les status des nœuds. Enfin, dans [VRG+14], les auteurs ont adapté le trickle timer pour gouverner l'envoie de messages DIO afin qu'ils transmettent les informations de routage sur la topologie de RPL à la 802.15.4 synchrone au niveau MAC.

## 4.7 Simulations et expérimentations

Il existe plusieurs implémentations du protocole RPL dans des simulateurs (OMNET++ WSNET et NS-2 et des systèmes d'exploitations ContikiOS, tinyOS, RIOT, OpenWSN). Les systèmes d'exploitations les plus importants et les plus utilisés sont TinyOS et ContikiOS. Par ailleurs, l'interoperabilité des deux implémentations a été comparée dans un travail [KET+11]. Durant cette thèse, nous avons travaillé avec le système d'exploitation Contiki et le simulateur Cooja. Nous allons à présent les décrire.

### Contiki

Contiki [DGV04] est un système d'exploitation dédié à des réseaux restreints tels que les réseaux de capteurs. Il fournit une librairie uIP : une implémentation réduite de TCP/IP pour rendre le système d'exploitation adapté aux capteurs et plus généralement à des dispositifs aux ressources limitées. En effet, en plus de gérer les standards habituels UDP, TCP et HTTP, Contiki gère aussi les standards IEEE 802.15.4, 6LoWPAN, RPL et CoAP. De plus, Contiki est implémenté en C et est portable dans plusieurs plateformes. Quand au protocole RPL, il a été presque entièrement implémenté sur Contiki. Le nombre d'instances, le trickle timer, les messages de contrôle sont déjà implémentés, seul le mode non storing n'a jamais été implémenté.

### Le simulateur Cooja

Cooja [EÖF+09] est un simulateur de réseaux dédié aux réseaux de capteurs sans fils. Il offre la possibilité de faire des simulations avec plusieurs types de capteurs avec plateformes MicaZ, ZI, Sky ainsi que d'autres plateformes. Il permet l'émulation de plusieurs capteurs sur lesquels seront chargés un système d'exploitation (éventuellement Contiki) et

des applications. Les connexions réseau et les interactions avec les capteurs seront ensuite simulées par Cooja.

#### Plateforme FIT Iot-lab

*FIT* (Future Internet of the Things) est un projet développé par un consortium de cinq établissements de l'enseignement supérieur et de la recherche (Université Pierre et Marie Curie, Inria, Université de Strasbourg, Institut Mines Télécom et CNRS). FIT a permis de développer des plateformes de capteurs sans fils fixes et mobiles, les interfaces qui permettent aux plateformes d'opérer dans une fédération et les outils nécessaires pour permettre aux utilisateurs d'accéder aux plateformes. *Iot-lab* [FMNA15] est une plateforme développée au sein de FIT conçue pour faire des expériences avec des réseaux de capteurs sans fils à grande échelle. Son principal objectif est d'offrir un outil scientifique à accès ouvert multi-utilisateurs pour permettre la conception, le développement, l'optimisation et l'expérimentation liés à des objects intelligents. Iot-lab propose plus de 2700 nœuds répartis sur 6 plateformes dans six différents sites en France : Inria Grenoble (928), Inria Lille (640), ICube Strasbourg (400), Inria Saclay(307), Inria Rennes (256) et Institut Mines-Télécom Paris (160). En plus de ces 6 sites, un nouveau site à Lyon avec 29 nœuds a été récemment ouvert. Io-lab supporte différents types de nœuds (WSN430, ARM Cortex M3 et ARM A8) avec différentes architectures de processeurs (MSP430, STM32 et Cortex-A8) et différentes puces sans fils (802.15.4 physique avec une fréquence de 800 MHz ou de 2,4 GHz). Chaque nœud comprend une passerelle qui permet la connexion à l'infrastructure mondiale Iot-lab pour scanner, contrôler et surveiller les nœuds à la fois lors de l'exécution et avant de lancer l'experimentation.

## 4.8 Conclusion

Dans ce chapitre, nous avons abordé les réseaux de capteurs. Nous avons présenté ces réseaux, leur caractéristiques et leurs contraintes. Nous avons ensuite donné un aperçu sur l'empilement des couches protocolaires dans de tels réseaux. Dans la section suivante, nous avons expliqué longuement le standard IEEE 802.15.4 et le protocole RPL. Nous avons aussi présenté un état de l'art sur le protocole RPL ainsi que quelques protocoles concurrents. Par ailleurs, nous avons expliqué que les évaluations de performances du protocole RPL ont montré qu'ils souffrait d'une instabilité de la topologie DODAG et d'une perte de paquets. Dans le prochain chapitre, nous allons présenter une des principales contributions de ma thèse qui propose une amélioration des performances de RPL.

# Chapitre 5

# Protocole RPL à degré borné

## 5.1 Introduction

Le chapitre précédent décrit l'ensemble des technologies étudiées durant la thèse : la IEEE 802.15.4 au niveau physique et MAC, le protocole RPL au niveau routage et la couche d'adaptation 6LoWPAN. Dans ce chapitre, nous allons considérer spécifiquement le protocole RPL dont le fonctionnement a été longuement décrit dans le chapitre précédent. Dans ce chapitre, nous proposons une solution pour améliorer les performances du protocole RPL. Nous rappelons que le protocole RPL a été proposé par le groupe ROLL de l'IETF. RPL est un protocole de routage IPv6 à vecteur de distance qui construit de façon proactive une topologie arborescente appelée *DODAG*. Le DODAG construit fournit les routes ascendantes des nœuds vers la racine et les routes descendantes de la racine vers les autres nœuds du réseau.

Les travaux [GK12, ITN13] ont souligné que la topologie construite par RPL n'est pas stable (Instabilité due aux réseau de capteurs), ce qui affecte les performances du protocole. C'est pourquoi nous proposons dans cette contribution un mécanisme qui fournit une meilleure stabilité à la topologie afin de réduire la perte de paquets, tout en améliorant le délai de transmission et en conservant une faible consommation énergétique. A notre connaissance, ceci est le premier travail, qui dans le but de réduire l'instabilité du DODAG construit par RPL, considère l'impact de la construction des routes descendantes sur les performances du protocole. En particulier, nous proposons de borner le nombre de routes descendantes qu'un nœud peut accepter, nous bornons ce nombre à une constante $k$ donnée. Le DODAG résultant est alors un arbre de degré $k$. Nous avons appelé notre solution *Bounded Degree RPL* (BD-RPL). Pour ce faire, nous utilisons les messages de contrôle standards fournis par le protocole RPL afin de ne pas générer une surcharge en messages de contrôle. BD-RPL produit une topologie réseau plus stable, en établissant une meilleure distribution de charge sur les nœuds. Le mécanisme de mise à jour implémenté dans RPL ne traite pas correctement la mise à jour des routes descendantes puisque celles-ci ne sont jamais supprimées de la mémoire. C'est pour cela que nous proposons dans BD-RPL un schéma d'échange de messages afin de supprimer les routes descendantes obsolètes. Les résultats de ce chapitre ont été publiés dans le workshop International Q2SWinet dans la conférence MSWiM en 2015 [BBLM15].

Notre avons implémenté BD-RPL dans Contiki. Nous avons ensuite évalué les performances de notre solution avec le simulateur Cooja et dans la plateforme Iot-lab. Nos résultats ont montré que BD-RPL fournit une meilleure stabilité de la topologie. Cela s'ex-

plique par le fait que les nœuds perdent moins de temps à s'adapter aux changements de la topologie. ce qui a pour résultat, un taux de délivrance de paquets plus élevé, un délai de transmission plus court et une consommation énergétique réduite. De plus, comme la topologie construite par BD-RPL est plus équilibrée, l'énergie consommée est mieux distribuée à travers les nœuds du réseau. La durée de vie du réseau est évaluée comme étant le temps durant lequel le réseau reste connecté, le meilleur équilibrage de charge au sein des nœuds, augmente la durée de vie du réseau.

**Organisation du chapitre** L'organisation de ce chapitre est la suivante, nous poserons tout d'abord la problématique en expliquant les causes de l'instabilité de la topologie. Nous présenterons par la suite les précédents travaux qui proposent des solutions à ce problème. Nous présenterons ensuite notre solution pour améliorer les performances du protocole RPL. Dans la section suivante, nous exposerons nos résultats de simulations et expérimentations. Enfin, dans le dernier chapitre, nous conclurons notre travail et donnerons des directions pour des travaux futurs.

## 5.2 Problématique

Comme nous l'avons vu dans le chapitre précédent, il existe plusieurs évaluations de performances du protocole RPL [TdOV10b, RSA12a, ITN13, AGBC11, GK12, CHP11]. Ces études cherchent à comprendre le comportement du protocole et à évaluer ses performances réseau. La plupart des ces études ont montré que parmi les protocoles de routage dédiés aux réseaux LLN, RPL offre les meilleures performances en termes de délai, d'énergie consommée et de surcharge en messages de contrôle. De plus, en terme de reception de paquets, RPL possède un taux de réception légèrement inférieur aux autres protocoles. Les auteurs dans [ITN13] ont aussi montré que RPL a une topologie instable à cause notamment d'une métrique de routage non optimale.

### 5.2.1 Impact de l'instabilité de la topologie d'un réseau de capteurs

La topologie instable du protocole RPL est due à plusieurs causes. En premier lieu, la nature des métriques ne prend pas en compte les interférences dues à la présence d'autres technologies comme des points d'accès 802.11.4 qui peuvent opérer sur la même bande de fréquence. De plus, à cause d'une métrique non optimale, un nœud $v$ peut décider de modifier son parent même si ce dernier est toujours accessible. Le nœud prend cette décision parce que la qualité du lien avec le parent, évaluée par la métrique, a diminuée. La mesure de baisse de qualité du lien est souvent due à des pertes de paquets entre le nœud et son parent. Cependant, dans ContikiMac, cela peut aussi être le résultat de la phase d'endormissement de l'interface radio des nœuds, introduite pour accroître l'efficacité énergétique dans la norme 802.15.4. Cela signifie que le nœud récepteur n'est potentiellement pas réveillé au moment ou les paquets lui sont transmis. Il est important de noter que le changement de parent a un effet domino. Lorsqu'un nœud $v$ change de parent, il choisit un parent avec une métrique potentiellement plus mauvaise que celle de son parent actuel. Cette nouvelle métrique est envoyée par $v$ vers ses descendants $w$. Si un nœud $w$ a un voisin disponible avec une meilleure métrique, $w$ changera alors son parent, entrainant éventuellement un changement en cascade depuis $v$ jusqu'aux feuilles de son sous-arbre. Tous ces changements entrainent des pertes de paquets, et donc de nouveaux changements de parent.

Pour comprendre l'impact de cette instabilité sur les performances de RPL, nous avons évalué les indicateurs de performances tels que le nombre de changements de parent et le nombre de paquets perdus. Pour observer ce comportement, nous avons fait 5 simulations préliminaires du protocole RPL dans le simulateur Cooja pour un réseau avec 50 nœuds. Les résultats sont représentés dans la Figure 5.1. Les indicateurs de performances sont en fonction du temps.

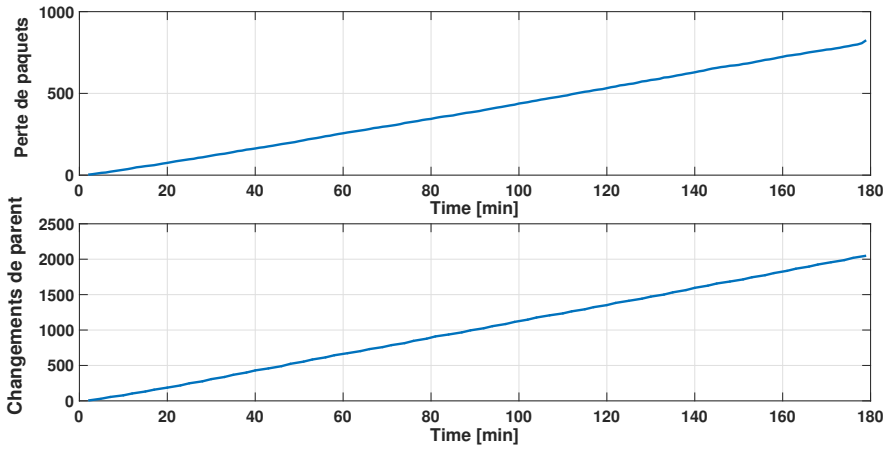



FIGURE 5.1 – Nombre de changements de parent et nombre de paquets perdus en fonction du temps de simulation.

Les résultats mettent en avant que le nombre de paquets perdus est proportionnel au nombre de changements de parent. En outre, nous observons que les deux courbes croissent durant toute la simulation, et ceci même après la convergence du DODAG, qui est obtenue au bout de quelques minutes de simulation. Les courbes mettent donc en évidence la forte relation entre l'instabilité de la topologie (DODAG) construite par RPL et la perte de paquets.

### 5.2.2 Impact d'un DODAG non borné

Dans RPL, la construction de la topologie ne prend pas en compte le nombre d'enfants par nœud, à savoir le degré du DODAG. Or, le nombre d'enfants a un impact sur la perte de paquets car les collisions de paquets envoyés à un nœud $v$ augmentent avec le nombre d'enfants de $v$. De plus, un nombre d'enfants élevé a des effets sur les performances du protocole en termes d'énergie consommée et de délai de transmission. L'énergie consommée par un nœud est due à la mémoire de stockage utilisée par le nœud, ainsi qu'au nombre de paquets transmis et reçus. En conséquence, plus grand est le nombre d'enfants du nœud, plus grande est l'occupation de la mémoire pour stocker les tables de routage de ses enfants, et plus grand est le nombre de messages envoyés et reçus. Par ailleurs, plus grand est le nombre d'enfants, plus grand est le délai de transmission car le nombre de collisions des paquets envoyés au parent augmente ralentissant de ce fait la transmission. Notre objectif principal est donc de contrôler la stabilité du DODAG en limitant le degré des nœuds. L'avantage de cette approche est une amélioration du taux de réception de paquets, un meilleur contrôle de la topologie et une meilleure répartition de la consommation énergétique sur les nœuds.

### 5.2.3 Améliorations de RPL

A notre connaissance, il n'y a qu'un seul travail qui traite de l'excès des routes descendantes [LHD13] dans le protocole RPL. Dans cet article, les auteurs considèrent ,entre autres, que le maintien des routes descendantes pour tous les nœuds est coûteux en termes de surcharge en messages de contrôle et en mémoire. Ils proposent d'utiliser, en fonction de besoins spécifiques, un schéma proactif ou réactif pour construire les routes descendantes. Leur approche réduit la surcharge en messages de contrôle et la consommation d'énergie. Dans cet article, contrairement à [LHD13], nous conservons entièrement la nature proactive du protocole RPL afin de maintenir le taux d'échange de messages de contrôle aussi bas que possible. En outre, une meilleure stabilité de l'arbre permettra des économies d'énergie considérables, aussi bien que la réduction du délai.

## 5.3 BD-RPL

Dans cette section, nous allons décrire le protocole BD-RPL. Le réseau est modélisé comme un graphe non orienté connexe $G(V, E)$. Nous définissons une constante $k < |V|$, où $|V|$ est le nombre de nœuds dans le graphe. Ce nombre constant $k$ est connu par chaque nœud et représente le nombre d'enfants qu'il peut accepter. En d'autres termes, $k$ est le degré du DODAG. Dans les réseaux de capteurs que nous considérons, la racine n'est pas limitée en batterie, nous avons donc choisi en conséquence de ne pas borner le nombre d'enfants de la racine.

Durant la construction de l'arbre couvrant (DODAG), chaque nœud $v$ dans le DODAG sélectionne un parent préféré $p$ et un ensemble de parents potentiels alternatifs pour construire les routes ascendantes. Après cela, $v$ envoie un DAO à $p$ afin de construire la route descendante. Notre solution pour borner le degré du DODAG intervient à ce niveau : chaque nœud $v$ dans le DODAG borne à $k$ le nombre d'enfants qu'il peut accepter. Cette limitation ne concerne que le nombre d'enfants, tandis que le nombre de descendants dans le sous-arbre n'est pas limité par BD-RPL.

Notre mécanisme utilise les messages de contrôle standards de RPL, tels que les messages DAO et DAO-ACK, afin de borner le degré des nœuds. Nous avons modifié les messages DAO-ACK en ajoutant un nouveau champ qui contient le refus ou l'acceptation par le parent. Par conséquent, notre solution se base sur les messages de contrôle existants fournis par RPL. Nous introduisons seulement une surcharge mineure quand un nœud reçoit un refus de son parent et envoie un message à un autre parent. Notre mécanisme exécuté par chaque nœud du DODAG excepté la racine, fonctionne comme suit :

- Dès qu'un nœud $v$ sélectionne son parent préféré $p$ dans le DODAG, il envoie un message DAO à $p$ pour établir la route descendante.
- Quand $p$ reçoit le message DAO, il vérifie le nombre d'enfants qui lui sont associés. Si cette valeur est inférieure à $k$, il accepte $v$ comme enfant et ajoute la route vers $v$ à sa table de routage. Ensuite, $p$ envoie un DAO-ACK à $v$ pour acquitter l'association. Cependant, si le nombre d'enfants a déjà atteint $k$, $p$ n'accepte pas $v$ en envoyant un DAO-ACK avec un refus à $v$.
- Lorsque $v$ reçoit le message DAO-ACK, si c'est un DAO-ACK avec une acceptation, $v$ considère la route ascendante pour $p$ comme établie et arrête l'envoi de messages DAO. D'autre part, si le message DAO-ACK contient un refus, $v$ choisit un nouveau parent $p'$ dans l'ensemble de ses parents successeurs et envoie un nouveau DAO à $p'$.

Le pseudo code de l'algorithme 6 présente le protocole BD-RPL.

**Algorithm 6:** BD-RPL pour un nœud $v$.

**Reception** d'un DIO envoyé par le nœud $u$ :
  $N(v) := \{u\} \cup Voisins(v)$
  **si** $rang(u) < rang(p_v)$ **alors**
    $parent_v := u$
    **Mise à jour** de $rang(v)$
    **Envoie** DAO à $u$
  **sinon**
    **si** $|sucesseur(v)| < B$ **alors**
      $successeur(v) := sucesseur(v) \cup \{u\}$
    **sinon**
      **si** $\exists w \in sucesseur(v) \mid rang(w) > rang(u)$ **alors**
        $successeur(v) := sucesseur(v) \cup \{u\} \setminus \{w\}$
  **Envoie** DIO à $N(v)$

**Reception** d'un DAO envoyé par le nœud $v$ :
**si** $Enfants(u) < k$ **alors**
    $Enfants(u) := Enfants(u) \cup \{v\}$
    **Envoie** DAO-ACK (1) à $v$
**sinon**
  **Envoie** DAO-ACK (0) à $v$

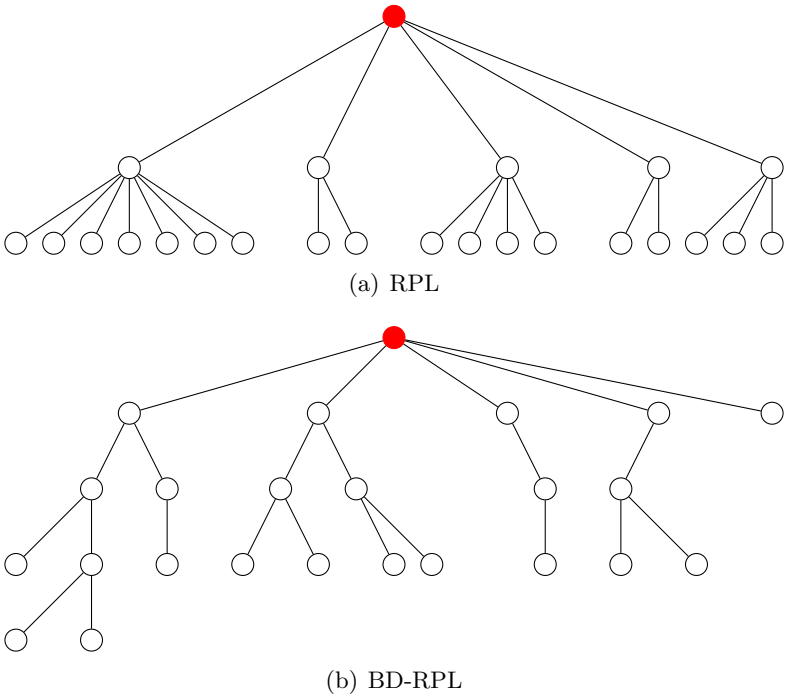
**Reception** d'un DAO-ACK envoyé par le nœud $u$ :
**si** $DAO - ACK(0)$ **alors**
    $parent_v := w$
    **Envoie** DAO à $parent_v$
**sinon**

Le problème de l'arbre couvrant à degré borné est un problème NP-complet [FR94]. Cependant, comme dans de nombreux réseaux LLN, le réseau est assez dense pour pouvoir converger vers un arbre couvrant de degré borné comme le démontre nos simulations et nos experimentations. Ainsi, les nœuds ont toujours un parent dans un DODAG de degré borné. Bien entendu, le mécanisme qui borne les routes descendantes affecte également les routes ascendantes. Nous rappelons qu'avec BD-RPL, le parent $p$ choisi par le nœud $v$ n'est pas nécessairement le voisin $u$ avec la meilleure métrique puisque que $u$ peut refusé $v$.

La Figure 5.2 illustre notre solution. Dans la Figure 5.2(a), où RPL est exécuté, le degré des nœuds dans le DODAG n'est pas borné. Une execution de BD-RPL est illustrée par la Figure 5.2(b). Dans ce cas, avec $k = 2$, le DODAG est plus équilibré, le nombre de collisions et de paquets perdus sera réduit, ce qui entraînera une économie d'énergie considérable. A noter que la profondeur du DODAG peut augmenter dans BD-RPL par rapport à RPL.

### 5.3.1 Impact de BD-RPL

Le protocole RPL ne fournit pas un mécanisme pour supprimer les routes descendantes non utilisées. Quand un parent ajoute une route descendante vers un descendant dans sa table de routage, cette route reste enregistrée même lorsque l'enfant change de parent.



(a) RPL

(b) BD-RPL

FIGURE 5.2 – DODAG RPL comparé à un DODAG BD-RPL.

Cette accumulation d'informations obsolète a un impact sur les performances du réseau.

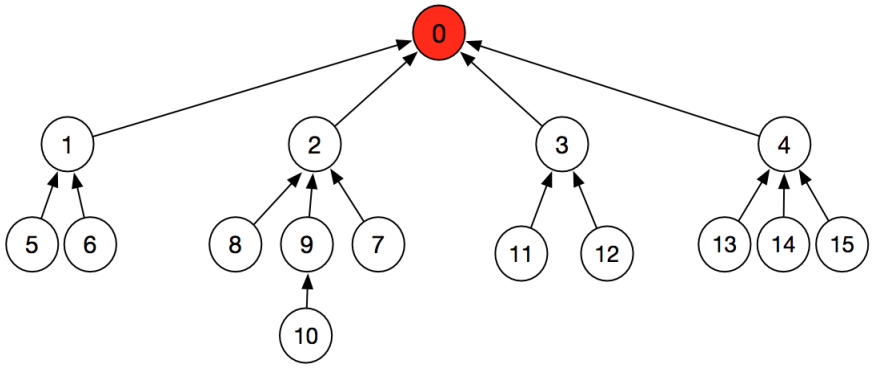
Premièrement, le nombre de routes descendantes augmente chaque fois qu'un nœud accepte un nouvel enfant. Cependant, les précédentes routes descendantes non utilisées ne sont pas supprimées. Ce nombre important de routes descendantes augmente la taille de la mémoire utilisée.

Deuxièmement, avec BD-RPL, lorsqu'un un nœud limite ses enfants à un degré maximum $k$, il enregistre uniquement les $k$ premiers enfants et leurs descendants. Ensuite, les nouveaux enfants qui arrivent avec les routes descendantes ne sont pas enregistrés car le nombre maximum d'enfants a déjà été atteint. Ainsi, seules les routes descendantes des $k$ premies enfants sont ajoutées. En pratique, avec BD-RPL, le nombre de changements de parent est réduit de manière significative, les nœuds conservent le même parent pendant une très longue période de temps. Ceci est essentiellement due au fait que d'autres parents potentiels peuvent les refuser car ils ont atteint la borne du nombre d'enfants.

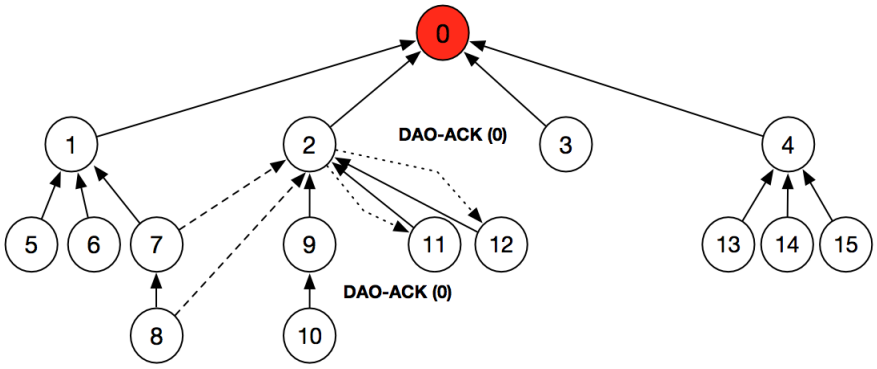
La Figure 5.3 donne un exemple de l'impact de BD-RPL sur un DODAG non mis à jour. Prenons la Figure 5.3(a) où le DODAG est borné. Les nœuds 7 et 8 ont tout d'abord le nœud 2 comme parent. Ensuite, dans la Figure 5.3(a), ils ont changé de parent (ils ont respectivement 1 et 7 comme parents). Comme RPL ne prévoit pas la mise à jour des routes descendantes, le nœud 2 considère qu'il a encore les nœuds 7 et 8 comme enfants. Par conséquent, lorsque les nœuds 11 et 12 prendrons le nœud 2 comme parent, ce dernier va les refuser (envoye un DAO-ACK (0)) 5.3(b).

### 5.3.2 Mise à jour des routes descendantes

Afin de résoudre les problèmes mentionnés ci-dessus, nous ajoutons dans BD-RPL la mise à jour des routes descendantes. BD-RPL permet aux nœuds parents de mettre à




(a) DODAG borné non mis à jour.



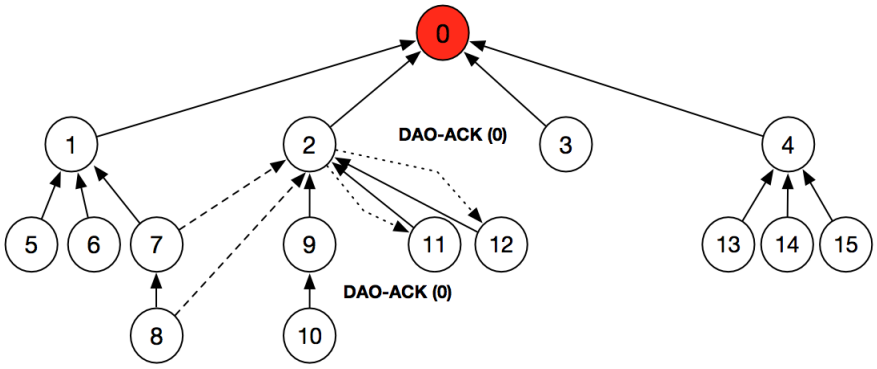

(b) changements de parent dans le DODAG borné.

FIGURE 5.3 – DODAG non mis à jour.

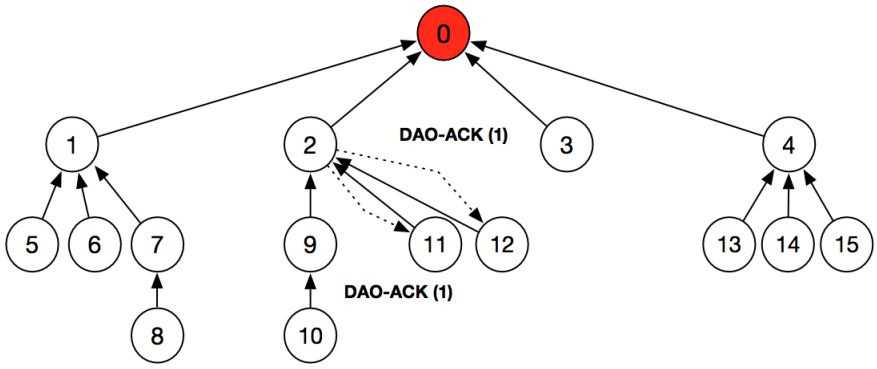
jour la liste de leurs enfants. À cet effet, on exploite les messages de contrôle DAO et DAO-ACK fournis par le protocole RPL pour signaler une route descendante non utilisée de la manière suivante :

- Quand un nœud $v$ change son parent préféré de $p$ à $p'$, il envoie un message DAO pour spécifier qu'une route descendante est non utilisée au parent $p$.
- Lorsque le nœud $p$ reçoit le message DAO, il supprime l'enfant $v$ et toutes les routes descendantes associées à $v$.

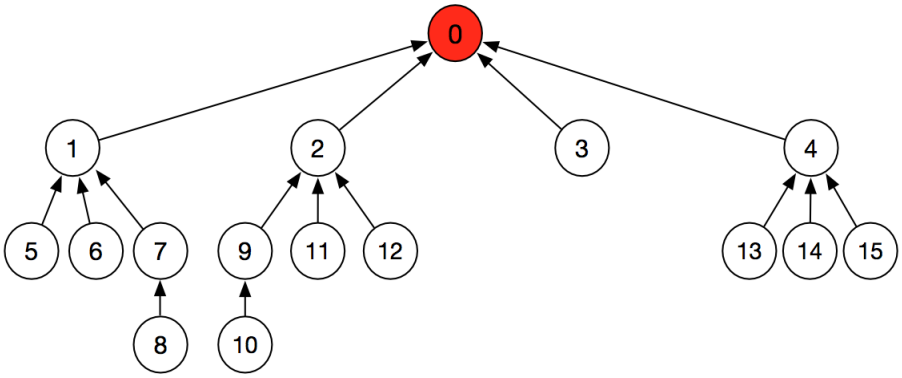
De cette façon, les nœuds suppriment les enfants obsolètes et disposent d'un espace libre pour ajouter de nouveaux enfants et les routes descendantes associées. Notre mécanisme améliore la délivrance de paquets dans le sens descendant, car les messages sont envoyés sur les routes mises à jour. En outre, notre mécanisme permet à la topologie d'évoluer avec les variations des liaisons radio. Les nœuds peuvent par conséquent changer de parent lorsque le lien vers le parent précédent devient mauvais. Enfin, notre mécanisme supprime toutes les routes descendantes non utilisées, réduisant ainsi l'utilisation de la mémoire. La Figure 5.4 est un exemple d'un DODAG borné mis à jour.




(a) Refus du parent dans un DODAG non mis à jour.




(b) Mise à jour des routes du parent.




(c) DODAG borné mis à jour

FIGURE 5.4 – DODAG mis à jour.

## 5.4 Implémentation de BD-RPL

### 5.4.1 Expérience personnelle

ContikiRPL est un système d'exploitation qui possède des centaines de milliers de lignes de code implémentées selon les spécifications de plusieurs *RFC* dont : RPL, IEEE 802.15.4, CoAP, ... qui sont tous sur plusieurs couches réseau (c'est-à-dire couche physique, couche mac, couche routage, couche application, ...). Le début de la thèse a été dédie à la maitrise de ces concepts et à l'étude des programmes afférents ainsi qu'a la maitrise du simulateur Cooja et de la plateforme Iot-lab.

Ajouté à ce travail de maitrise de plusieurs mois, nous avons aussi développé notre solution BD-RPL dans plusieurs fichiers du système d'exploitation Contiki. Plus précisément, nous avons implémenté BD-RPL dans les fichiers Contiki suivants : rpl.c, rpl-dag.c, uip-ds6-route.c, tcpip.c, rpl-icmp.c, udp-sender.c, udp-server.c, projetc-conf.h. Voici une description plus détaillée de cette implémentation dans ces fichiers.

— Dans le fichier **project-conf.h**, nous avons spécifié nos paramètres de simulations et d'expérimentations. Nous pouvons citer de façon non-exhaustive : le temps de convergence du DODAG (autrement dit le temps après lequel les nœuds peuvent envoyer de l'information), l'interval de transmission des données et leurs acquittements (activation du trafic descendant) et la fréquence de vérification du canal radio.

— Le fichier **udp-sender.c** est exécuté par tous les nœuds du DODAG à part la racine. Ce fichier fait appel au système d'exploitation Contiki avec tout l'empilement des couches, notamment celle de RPL. Nous avons inséré des compteurs (*timers* en anglais) pour l'envoi et la réception périodique des messages de données par les nœuds. De plus, nous avons mis en place dans ce fichier le calcul de l'énergie consommée par chaque nœud.

— Le fichier **udp-server.c** est exécuté uniquement par la racine du DODAG. Il possède les mêmes caractéristiques que le fichier udp-sender.c. La différence entre les deux fichier réside dans le fait que la racine n'envoie de messages (acquittements) qu'au moment où elle reçoit un message de l'un de ses enfants. Une autre différence entre les deux fichiers est que nous n'avons pas implémenté le calcul de l'énergie pour la racine car nous avons choisi un réseau de capteurs où la racine n'est pas limitée en énergie.

— Nous avons modifié le fichier **uip-ds6-route.c** afin qu'un parent qui ajoute ses nouveaux enfants puisse les comptabiliser.

— Dans le fichier **tcpip.c**, nous avons implémenté un compteur (*timer*) pour vérifier la présence d'un enfant. Lorsque le compteur expire et que le parent n'a reçu aucune donnée de son enfant, il considère qu'il l'a quitté et supprime l'enfant.

— Dans le fichier **rpl-icmp6.c**, nous avons modifié les messages de contrôle DAO et DAO-ACK. Plus précisément, nous avons modifié les messages DAO entrants afin qu'un parent lance la procédure de suppression d'une route (associée à un enfant en appelant une fonction que nous avons implémenté dans le fichier rpl.c) si ce message DAO spécifie que l'enfant l'a quitté. D'autre part, nous avons aussi modifié les messages DAO entrants afin que le nœud teste le nombre d'enfants qu'il a et décide en conséquence (d'ajouter ou non la route descendante associée à l'enfant). Par ailleurs, nous avons aussi modifié le message DAO-ACK sortant (que le parent envoie) afin que le parent examine son nombre d'enfants et spécifie, avant d'envoyer

le message, si l'enfant est accepté ou non. Dans notre message DAO-ACK sortant modifié, un enfant teste s'il est accepté par son parent ou non. S'il est accepté, il n'envoie aucun autre message de contrôle. Sinon, l'enfant supprime son parent, ajoute un nouveau parent et envoie un nouveau message à son nouveau parent.

— Dans le fichier **rpl.c**, nous avons inséré la suppression des anciens enfants des nœuds.

— Dans le fichier **rpl-dag.**c, nous avons modifié le changement et la suppression du parent.

Avec le simulateur Cooja, en plus de travailler sur les fenêtres graphiques fournies pas le simulateur, nous avons manipulé des fichiers xml et des fichiers Makefile pour automatiser l'exploitation des résultats. Pour traiter les fichiers de résultats (log) et générer des courbes, nous avons écrit des scripts en sh et en python et traité nos données avec Matlab. Ce travail a aussi nécessité plusieurs milliers de lignes de code.

Sur la plateforme Iot-lab, nous avons tout d'abord travaillé sur les nœuds capteurs : interaction avec les microprogrammes s'executant sur les nœuds, manipulations et communications avec les interfaces des nœuds, lecture des valeurs des capteurs et récupération de données radio. Ensuite, afin de récupérer des fichiers de résultats (log), nous avons assigné des identifiants aux nœuds de la plateforme, introduit le temps d'expérimentation et écrit les scripts nécessaires pour traiter les résultats retournés par les nœuds.

Comme nous avons validé nos travaux par simulations et expérimentations, une grande partie du travail de cette thèse a porté sur la maitrise d'outils de simulations (le simulateur Cooja) et d'expérimentations (la plateforme Iot-lab). Cet effort fournit et la connaissance de la plateforme Iot-lab ont conduit à une collaboration avec d'autres membres de l'équipe NPA du laboratoire lip6 pour présenter une démo et un poster lors de la conférence MobiCom en Septembre 2015 [BBK+15].

Pour plus de détails techniques sur le code BD-RPL et les outils utilisés (Iot-lab, Cooja, Matlab, scripts sh et Python), voir ma page web.

### 5.4.2 Implémentation

Dans cette section, nous évaluons le protocole BD-RPL avec le degré du DODAG borné à 3 sur une implementation sur Contiki. En premier lieu, nous avons fait des simulations sur le simulateur Cooja. Ensuite, nous avons validé nos résultats par des expérimentations sur la plateforme Iot-lab. A noter que nous ne comparons pas les performances de la racine puisque avec BD-RPL, la racine ne borne pas ses enfants. Nous comparons les performances de BD-RPL avec RPL en termes de :

- **Stabilité de la topologie :** nombre de changements de parent par nœud.
- **Consommation de l'énergie Radio :** estimation de l'énergie radio consommée pour envoyer et recevoir des messages.
- **Délai de transmission :** temps que prend un paquet pour atteindre sa destination. Dans la simulation, nous calculons le délai moyen de la transmission dans le sens ascendant à partir des nœuds vers la racine et dans le sens descendant à partir de la racine vers les autres nœuds. Comme dans l'expérimentation, les horloges des nœuds ne sont pas synchronisées, nous considérons le délai de bout en bout, calculé pour chaque nœud, comme le temps nécessaire pour envoyer un paquet et recevoir son accusé de réception.

• **Taux de délivrance de paquets :** c'est le rapport entre le nombre de paquets reçus et le nombre de paquets envoyés. Cela indique la fiabilité du protocole de routage.

### 5.4.3 Configuration

Nous avons implémenté BD-RPL sur *Contiki* [DGV04], un système d'exploitation (open source) dédié aux réseaux LLN contraints. Contiki est conçu pour utiliser une petite quantité de mémoire tout en supportant au niveau réseau un empilement IP complet avec le protocole RPL ainsi qu'une couche d'adaptation 6LoWPAN. Contiki supporte également ContikiMAC [Dun11], un mécanisme radio avec un duty cycle dans lequel les nœuds dorment la majorité du temps et se réveillent périodiquement afin de conserver l'énergie consommée pendant l'écoute et la transmission. ContikiMAC s'exécute au-dessus de l'IEEE 802.15.4 au niveau de la couche physique. La métrique de routage utilisée dans le protocole RPL est la métrique ETX [CABM03]. Tous les nœuds du réseau sont configurés en mode storing. Nous supposons que les nœuds ont une puissance de transmission élevée. En conséquence, chaque nœud a plusieurs voisins et un large choix de parent. L'ensemble des éléments de la configuration choisis sont résumés dans le tableau 5.4.3.

| **Élément** | **Configuration** | **Description** |
|---|---|---|
| Système d'exploitation | Contiki | Dédié aux réseaux LLN |
| Couche physique | IEEE 802.15.4 | Antenne radio |
| Couche MAC | ContikiMAC | Duty cycle |
| Couche routage | RPL/BD-RPL | Mode Storing |
| Métrique de routage | ETX | Qualité du lien radio |

FIGURE 5.5 – Tableau résumant les éléments de la configuration choisis.

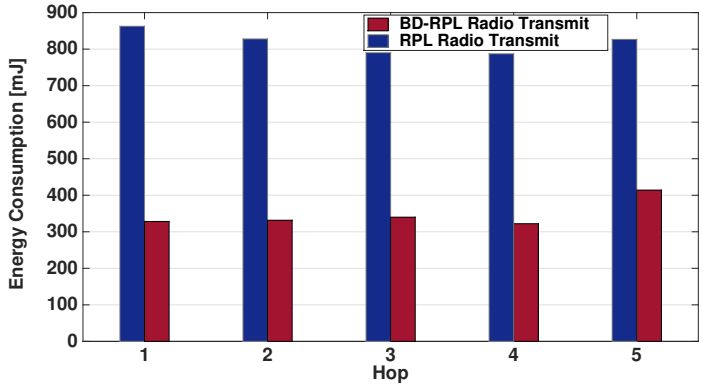
### 5.4.4 Simulations

Nous faisons nos simulation dans COOJA [EÖF+09], un simulateur pour réseaux de capteurs sans fil. COOJA simule chaque nœud comme une plate-forme TMOTE Sky basée sur un micro-contrôleur MSP430 à 16-bit et une interface radio à 2.4 GHz. Le taux , de vérification du canal radio est à 4 Hz. Dans notre configuration, nous considérons 50 nœuds déployés aléatoirement qui forment un réseau entièrement connecté. Le modèle de propagation est un Unit Disk Graph Model avec une portée de transmission et une portée d'interférence de 30 mètres avec un milieu de transmission sans perte. Les nœuds commencent la transmission de paquets vers la racine au bout de 2 minutes de simulation afin de laisser le temps au DODAG d'être construit. Après cette période de construction, les nœuds commencent à envoyer des paquets UDP vers la racine toutes les 2 minutes. En outre, la racine acquitte les transmissions réussies par des paquets toutes les 3 minutes. Chaque simulation est exécutée 5 fois pour réduire les fluctuations statistiques et chaque simulation dure 5 heures.

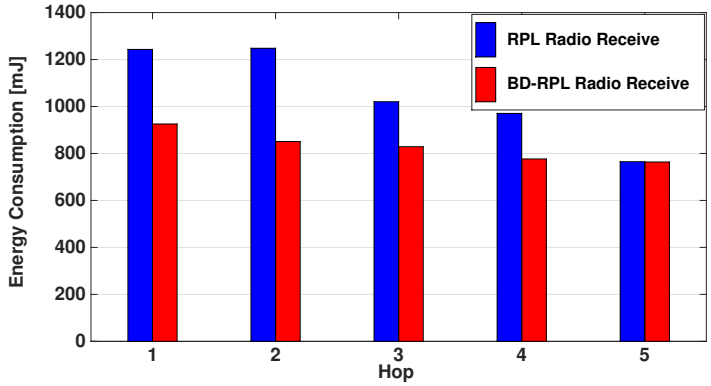
#### Résultats pour la consommation d'énergie

La Figure 5.6 présente la consommation moyenne de l'énergie radio en fonction de la distance en nombre de sauts jusqu'à la racine pour RPL et BD-RPL. La consommation

d'énergie est réduite avec BD-RPL durant les deux phases d'écoute et de réveil. En particulier, la consommation d'énergie durant la phase d'écoute est réduite puisqu'un nœud a moins d'enfants associés, tandis que la consommation d'énergie dans la phase d'émission est réduite car il y a moins de collisions pour transmettre au parent. L'avantage de BD-RPL par rapport à RPL est particulièrement important pour les nœuds les plus près de la racine. En outre, à partir de la Figure 5.6, on voit que l'énergie consommée dans BD-RPL est mieux répartie parmi les nœuds du réseau, ce qui entraîne une consommation plus équitable de l'énergie et une plus longue durée de vie du réseau.




(a) Énergie radio à la transmission.




(b) Énergie radio à la réception.

FIGURE 5.6 – Consommation de l'énergie radio en fonction de la distance en nombre de sauts.

### Résultats pour le délai de transmission

La Figure 5.7 présente le délai de transmission ascendant en fonction de la distance en nombre de sauts à la racine pour RPL et BD-RPL. La charge moyenne sur les nœuds est réduite, une conséquence directe au fait que le délai ascendant est réduit. Par conséquent, les paquets subissent moins de collisions et attendent un faible intervalle de temps avant d'être insérés dans le support sans fil. Comme les nœuds ont un nombre limité d'enfants, la transmission au parent devient moins compétitive et le délai de transmission est réduit.

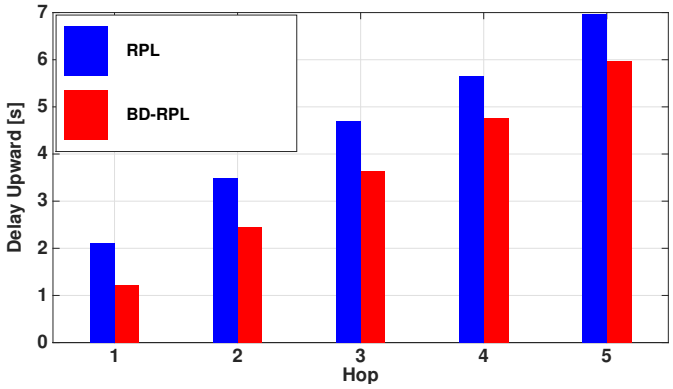



FIGURE 5.7 – Délai de transmission ascendant en fonction de la distance en nombre de sauts.

Dans la Figure 5.8, nous évaluons le délai descendant en fonction de la distance en sauts à la racine pour RPL et BD-RPL. Avec BD-RPL, le délai descendant est comparable à RPL pour les nœuds à proximité de la racine (jusqu'à 3 sauts). Par contre, pour les nœuds les plus profonds dans l'arbre, BD-RLP indique une meilleure performance du délai descendant car le nombre de collisions est plus faible avec BD-RPL.

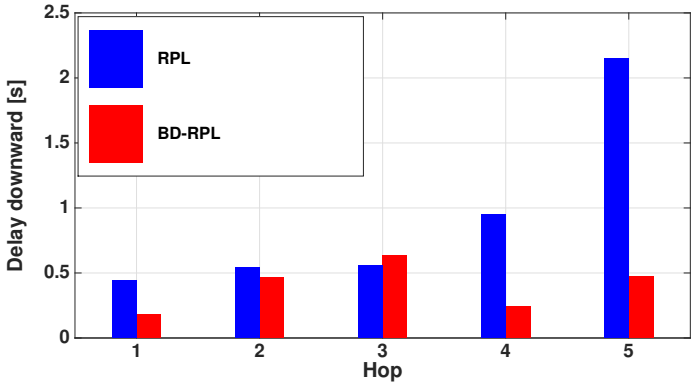



FIGURE 5.8 – Délai de transmission descendant en fonction de la distance en nombre de sauts.

### Résultats pour le taux de délivrance de paquets

Figure 5.9 montre le taux de livraison de paquets en fonction de la distance en sauts à la racine pour RPL et BD-RPL. Avec BD-RPL, le taux de réception de paquets est augmenté de près de 10% par rapport à RPL. Comme pour le délai de transmission,

réduire la charge sur les nœuds mène à une plus faible perte de paquets en raison de l'indisponibilité des parents. En effet, comme le protocole CSMA/CA s'execute sur les nœuds, après un certain nombre de tentatives infructueuses de transmission, le paquet est supprimé. Si la charge sur le nœud qui reçoit est faible, ce sera probablement valable pour la réception de paquets, le taux de réception de paquets augmentera en conséquence.

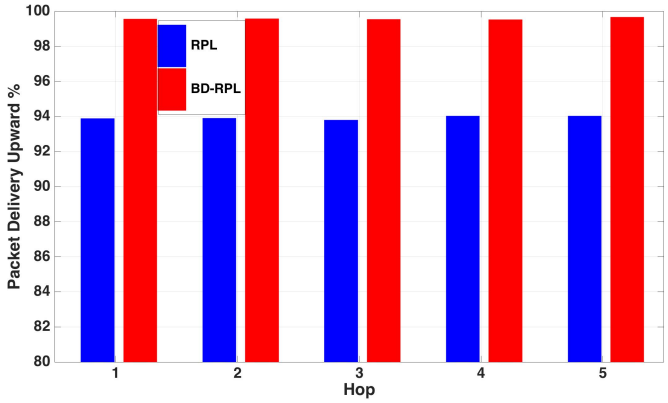



FIGURE 5.9 – Taux de délivrance de paquets ascendant en fonction de la distance en nombre de sauts.

### 5.4.5 Expérimentations

Cette section présente les résultats de BD-RPL avec la plateforme Iot-lab. Afin d'éviter la répétition, nous allons exclusivement souligner les différences entre les simulations et les expérimentations.

#### Configuration et hypothèses

Pour notre expérimentation de BD-RPL dans Iot-lab, nous avons considéré 50 nœuds WSN430 statiques. Dans nos expériences, nous avons considéré une radio CC2420, fonctionnant à 2,4 GHz. Le taux de vérification du canal est fixée à 4 Hz. La puissance d'émission des nœuds est fixé à -1 dBm. Les nœuds commencent à transmettre leurs paquets de données au bout de 2 minutes pour laisser le temps au protocole de construire l'arbre. Les nœuds envoient des paquets UDP toutes les deux minutes vers la racine. La racine acquitte chaque paquet de données reçu. Nous executions chaque experimentation 3 heures et nous la répétons 6 fois avec la même distribution de nœuds. Le tableau 5.4.5 résume les choix de configuration.

#### Résultats pour le changement de topologie

La Figure 5.11 compare le changement moyen de parents en fonction de la distance en nombre de sauts au puits. Les résultats montrent qu'avec BD-RPL, nous réduisons en moyenne le nombre de changements de parent par deux. Nous voyons sur cette figure que, avec BD-RPL, les nœuds qui sont à un saut de la racine réduisent de manière significative leur changement de parents. Cela est dû au refus de potentiels nouveaux parents, les forçant à rester avec leurs parents actuels. Cela confirme notre première intuition que l'excès de

| Implémentation | Configuration | Déscription |
|---|---|---|
| Plateforme | Iot-lab | Conçue pour faire des expérimentation WSN à large échelle |
| Nœuds WSN | WSN430 | Nœuds statiques 50 nœuds |
| Communication radio | CC2420 | Opère à $2.4GHZ$ |
| Radio | $4GHZ$ | Taux de vérification du canal toutes les 250 ms |
| Puissance de transmission | $-1dBm$ | Haute puissance de transmission Topologie dense |
| Trafic | 1 paquet/2 min | Paquet envoyé et acquitté |
| Temps | 3 heures | Répétée 6 fois |

FIGURE 5.10 – Tableau résumant les choix de configuration pour l'expérimentation.

changements de parent n'est pas toujours un bon comportement pour les performances du protocole.

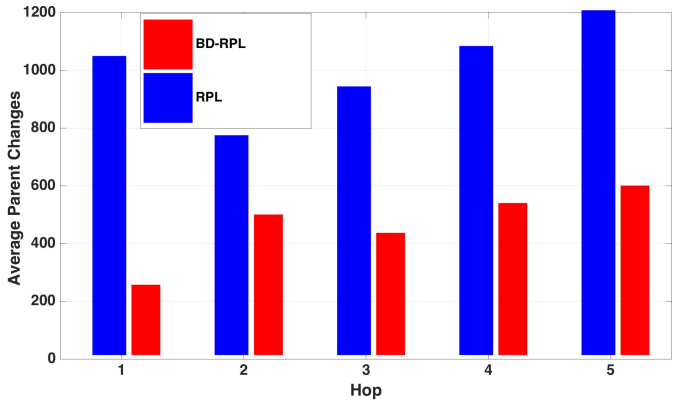



FIGURE 5.11 – Nombre moyen de changements de parent en fonction de la distance en nombre de sauts.

La Figure 5.12 représente le nombre de changements de parent en fonction du temps. Nous faisons la même analyse que pour le taux moyen de changements de parent en fonction du nombre de sauts. Nous faisons remarquer en plus que plus nous avançons dans l'expérimentation, plus le nombre de changements de parents pour BD-RPL est réduit par rapport à RPL.

### Résultats sur la consommation d'énergie

La Figure 5.13 représente l'énergie consommée par les nœuds pour envoyer et recevoir des paquets en fonction de la distance en sauts jusqu'à la racine pour RPL et BD-RPL. Dans l'expérimentation, la consommation d'énergie est 10 fois plus élevée que dans la simulation pour RPL et BD-RPL. En outre, comme cela a déjà été montré dans la Figure 5.13, la consommation d'énergie dans BD-RPL est réduite d'un facteur 2 par rapport à RPL. On remarque que les nœuds près de la racine réduisent leur consommation énergétique à

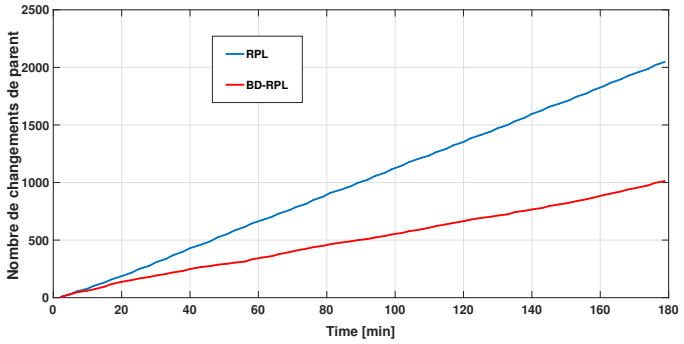
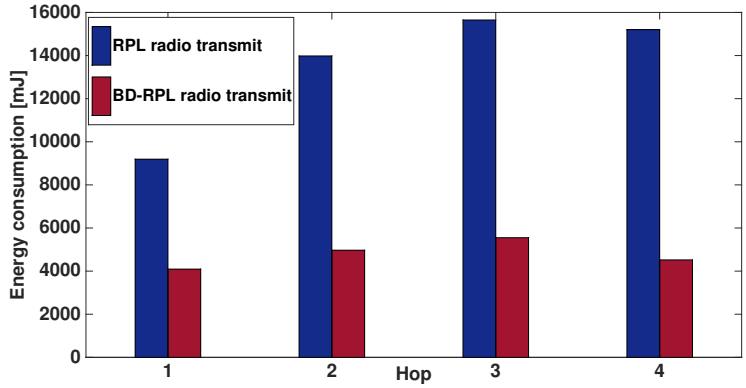
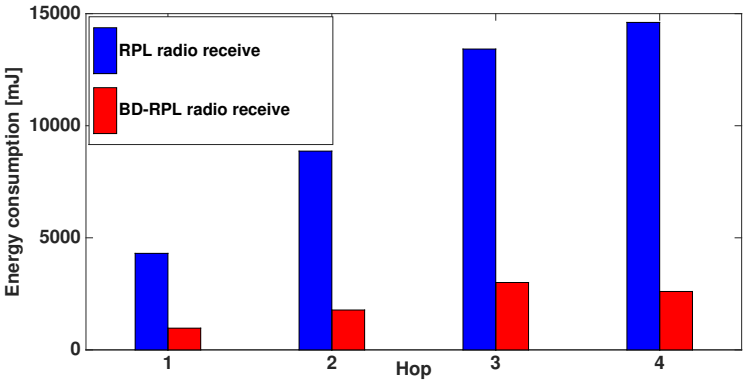



FIGURE 5.12 – Nombre de changements de parent en fonction du temps.

l'écoute parce qu'ils ont moins d'enfants à gérer. La consommation d'énergie dans la phase d'émission est elle aussi réduite car il y a moins de collisions entre les messages transmis au parent.




(a) Énergie radio à la transmission.




(b) Énergie radio à la réception.

FIGURE 5.13 – Consommation de l'énergie radio en fonction du nombre de sauts.

### Résultats pour le délai de transmission

La Figure 5.14 représente le délai moyen de transmission de bout en bout en fonction de la distance en sauts jusqu'à la racine pour RPL et BD-RPL. Cette figure montre que,

comme avec la simulation, BD-RPL réduit le délai de bout en bout par rapport à RPL. Cet effet est plus évident pour les nœuds qui sont proches de la racine, car ils ont moins d'enfants à gérer et les paquets sont envoyés sans délai d'attente supplémentaire.

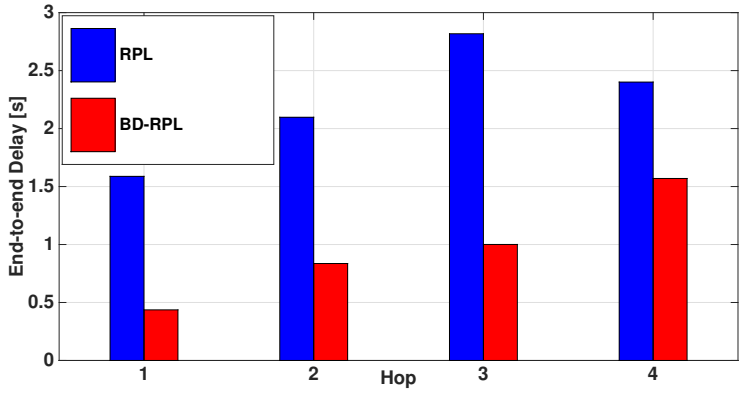



FIGURE 5.14 – Délai de transmission en fonction du nombre de sauts.

### Résultats pour le taux de délivrance de paquets

Les résultats dans la Figure 5.15 montrent qu'en expérimentations, nous améliorions le rapport de taux de délivrance de paquets ascendants. Ce qui est encore plus important que dans les simulations puisque l'amélioration est d'environ 10%. Cependant, nous reconnaissons que la perte de paquets dans des expérimentations réelles est plus importante que la perte de paquets en simulations en raison d'interférences du monde extérieur qui ne peuvent être capturées par des simulations.

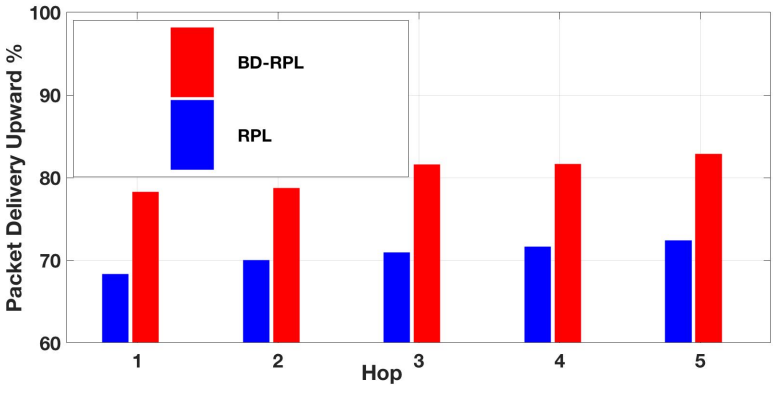



FIGURE 5.15 – Taux de délivrance de paquets en fonction du nombre de sauts.

La Figure 5.16 représente le taux de délivrance de paquets en fonction du temps d'expérimentation pour RPL et BD-RPL. Nous remarquons dans cette figure que le taux de délivrance de paquets est pratiquement similaire en début d'expérimentation. Ensuite, l'écart se confirme avec le temps d'expérimentation.

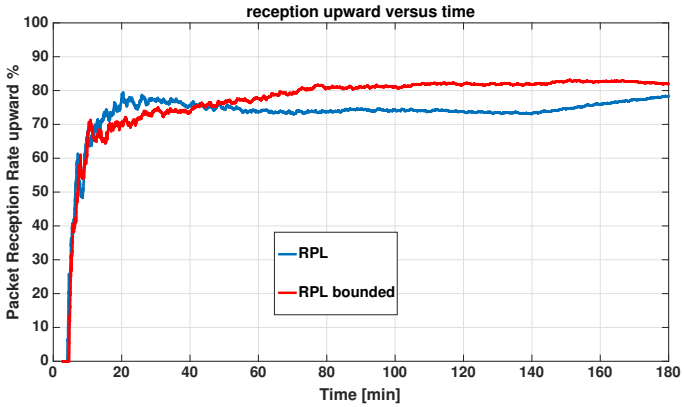



FIGURE 5.16 – Taux de délivrance de paquets en fonction du temps.

## 5.5 Conclusion

Dans ce chapitre, nous avons proposé un nouveau protocole : BD-RPL, une solution pour améliorer le protocole RPL. Le protocole BD-RPL considère l'instabilité des routes dans les réseaux LLN en introduisant une borne sur le nombre maximum d'enfants qu'un nœud peut accepter pendant la construction de l'arbre. De plus, BD-RPL traite l'absence de mise à jour sur les routes descendantes. En particulier, nous utilisons les messages de contrôle existants fournis par RPL pour limiter le degré des nœuds et pour mettre à jour les routes descendantes. Par conséquent, BD-RPL ajoute un taux de messages de contrôle marginal par rapport à RPL. Nous avons évalué BD-RPL en utilisant à la fois le simulateur Cooja et la plate-forme Iot-lab. Les simulations et les expérimentations ont prouvé une amélioration par rapport à RPL d'une moyenne de 10% dans le taux de délivrance de paquets, de 50% dans la consommation énergétique et de 60% en délai de transmission.

Ce travail met en évidence la question suivante. Nous avons vu l'absence de mécanisme de synchronisation de la communication entre les noeuds au niveau de la couche MAC (puisque nous utilisons la IEEE 802.15.4 non synchronisée). D'autre part, il est bien connu que dans un réseau de capteurs, les performances d'un protocole de routage dépendent à la fois de la couche réseau et de la couche MAC. Une perspective naturelle de ce travail serait donc une solution inter-couches entre la couche MAC et la couche réseau (ContikiMAC et RPL) afin d'introduire un mécanisme pour organiser la transmission des données des enfants vers leurs parents et par conséquent réduire les collisions et l'énergie consommée. Une façon de faire serait d'utiliser le même mécanisme que notre solution BD-RPL en ajoutant de la synchronisation.

# Chapitre 6

# Conclusion

La quantité de données échangées dans les réseaux d'aujourd'hui, la diversité des objets qui y sont connectés (ordinateurs, tablettes, téléphones, robots, capteurs ou tout objet connecté) et la grande taille des réseaux sont autant de raisons qui conduisent naturellement à des études sur les protocoles de routage auto-stabilisants. D'autant plus que ces protocoles de routages sont devenus de plus en plus exigeants : support de la dynamicité, faible consommation énergétique, passage à l'échelle, etc.

Les caractéristiques des réseaux de capteurs sont une faible portée de communication, une puissance de calcul limitée, une faible capacité de mémoire de stockage, et le plus souvent une batterie à durée de vie limitée. Les caractéristiques physiques intrinsèques de ces réseaux conduisent à de forts changements de topologie (disparition des capteurs, variabilité des liens radio) du réseau et ceci tout au long de la durée de vie de ce réseau. Ces changements de topologie peuvent entrainer des pertes d'informations.

C'est pourquoi L'auto-stabilisation [Dij74] est l'une des techniques adaptée pour maintenir la disponibilité des applications réseaux distribuées, leur fiabilité et leur maintenance. Après l'apparition d'une défaillance qui a placé les composants du réseau dans un état global arbitraire, l'auto-stabilisation garantit la récupération vers un comportement légitime en un temps fini sans aucune intervention extérieure.

Dans ce contexte, une tâche essentielle du réseau est de récupérer des communications efficaces. Une façon naturelle pour faire face à ce problème consiste à construire un arbre couvrant le réseau et d'acheminer les messages entre les nœuds grâce à cette structure, qui a l'avantage de fournir une et une seule route entre n'importe quelle paire de nœuds du réseau. Dans un réseau, il n'existe pas un unique arbre couvrant. En fonction des contraintes environnementales, l'arbre couvrant proposé peut optimiser des métrique différentes. Il existe ainsi, autant de protocoles de construction d'arbres couvrants sous-contraintes que d'arbres couvrants optimisant une certaine métrique.

Pour cette raison, il est non seulement important de construire des protocoles de routages qui prennent en compte les exigences de tels réseaux mais aussi de construire des algorithmes auto-stabilisants qui minimisent l'utilisation de la mémoire et qui garantissent une convergence rapide de la structure couvrante. Cette thèse a proposé deux contributions dans ce cadre.

## 6.1 Résumé des contributions de la thèse

**Arbre couvrant de diamètre minimum** Le diamètre d'un réseau est l'une des métriques réseau la plus fondamentale. Calculer le diamètre est un problème important dans l'analyse de grands réseaux. De plus, ce paramètre a de nombreuses applications pratiques importantes dans des réseaux réels. En effet, un arbre couvrant de diamètre minimum (MDiamST) est une approche naturelle si l'on veut optimiser le délai de communication entre toutes paires de nœuds dans un réseau, puisque la distance entre une paire de nœuds est limitée par le diamètre de l'arbre, qui est minimal dans le cas du MDiamST. Par conséquent, il est naturel d'étudier ce problème dans un réseau distribué, et plus particulièrement dans un réseau distribué tolérant aux fautes transitoires. C'est pourquoi dans la première contribution, nous avons proposé le premier algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum qui tolère tout environnement asynchrone. Autrement dit, nous considérons un démon distribué non équitable. Notre algorithme utilise une mémoire de $O(\log n)$ bits par nœud, ce qui améliore le résultat précédent [BLB95] d'un facteur $n$. De plus, notre algorithme se stabilise au bout de $O(n^2)$ rondes, qui reste polynomiale comparé aux résultats précédents.

**DODAG RPL borné** Le protocole RPL souffre de l'instabilité de sa topologie DODAG et par conséquent d'un signifiant taux de perte de paquets. C'est pourquoi nous avons proposé dans la deuxième contribution le protocole BD-RPL, une solution pour améliorer le protocole RPL. Cette modification du protocole RPL réduit l'instabilité des routes dans les réseaux de capteurs en introduisant une borne sur le nombre maximum d'enfants qu'un nœud peut accepter durant la construction du DODAG. De plus, nous avons identifié un problème d'implémentation dans la mise à jour des routes dans RPL et avons implémenté un mécanisme de mise à jour des routes descendantes. D'autre part, nous utilisons les messages de contrôle existants fournis par RPL pour borner le DODAG, ainsi que pour mettre à jour les routes descendantes. Par conséquent, BD-RPL ajoute un taux de messages de contrôle marginal par rapport à RPL. De surcroit, comme BD-RPL est indépendant de la métrique de routage, toute amélioration de la métrique utilisée par RPL ajoute des améliorations à DB-RPL. Enfin, nous avons évalué BD-RPL en utilisant à la fois avec le simulateur Cooja et la plate-forme Iot-lab. Les résultats d'expérimentations ont prouvé une amélioration par rapport à RPL d'une moyenne de 10% dans le taux de délivrance de paquets, de 50% dans la consommation énergétique et de 60% en délai de transmission.

## 6.2 Perspectives

Les contributions de cette thèse ouvrent plusieurs directions de recherche :

**Mémoire minimum pour un arbre couvrant de diamètre minimum.** Notre contribution sur la construction d'un arbre couvrant de diamètre minimum met en évidence les questions ouvertes suivantes. La première concerne l'optimalité en espace mémoire, autrement dit l'algorithme auto-stabilisant non-silencieux que nous avons proposé est-il optimal en mémoire ? En effet, des travaux [BGJ99] ont montré qu'il n'était pas possible de faire une construction auto-stabilisante non-silencieuse d'arbre couvrant en utilisant une mémoire constante. Mais d'autres résultats plus récents [BT13] ont montré qu'il était possible de faire une telle construction en utilisant une mémoire sous-logarithmique,

$O(\log \log n)$ bits par nœud pour être précis. Pouvons-nous appliquer de tels résultats où le problème de l'arbre couvrant de diamètre minimum nécessite une mémoire de $\Omega(\log n)$ et ceci même dans le cadre d'algorithmes non silencieux ?

La deuxième question concerne l'optimalité de l'espace mémoire des algorithmes silencieux dédiés au problème de l'arbre couvrant de diamètre minimum. Existe-il un algorithme silencieux pour la construction d'un arbre couvrant de diamètre minimum qui nécessite uniquement $O(\log n)$ bits de mémoire par nœud ? Une réponse à cette question est suggérée dans [BFP14]. En utilisant leur solution, la question est réduite à fournir un schéma d'étiquetage pour ce problème nécessitant $O(\log n)$ bits de mémoire par nœud. Si un tel schéma d'étiquetage existe, une adaptation pure et simple de notre algorithme auto-stabilisant serait un algorithme auto-stabilisant silencieux et optimal pour la construction d'un arbre couvrant de diamètre minimum.

**Implémentation d'un arbre couvrant de diamètre minimum dans un réseau de capteurs.** Nous avons proposé un algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum. Nous savons que dans un réseau de capteurs, réduire le délai de communication entre tous les nœuds du réseau est un défi important. Dans la plupart des cas, une racine dans un réseau de capteurs se place à la frontière du réseau. Dans le cadre de réseaux mobiles, nous pourrions nous inspirer de notre algorithme pour calculer un centre du réseaux et faire en sorte que la racine se déplace en ce centre, ce dernier s'adaptant au cours de la vie du réseau.

**Synchronisation de la communication dans le protocol RPL.** Nous avons vu dans ce manuscrit l'absence de mécanismes de synchronisation de la communication entre les nœuds au niveau de la couche MAC (puisque nous utilisons la IEEE 802.15.4 non synchronisée). D'autre part, il est bien connu que dans un réseau de capteurs, les performances d'un protocole de routage dépendent à la fois de la couche réseau et de la couche MAC. La dernière perspective serait donc une solution inter-couches entre la couche MAC et la couche réseau (protocole ContikiMAC et protocole RPL) afin d'introduire un mécanisme pour organiser la transmission des données des enfants vers leurs parents et par conséquent réduire les collisions et l'énergie consommée. Une réponse à cette perspective serait d'utiliser le même mécanisme que notre solution BD-RPL en ajoutant de la synchronisation.

# Bibliographie

[ABB98] Y. Afek and A. Bremler-Barr. Self-stabilizing unidirectional network algorithms by power supply. *Chicago J. Theor. Comput. Sci.*, 1998, 1998.

[AG90] A. Arora and M. Gouda. Distributed reset (extended abstract). In *Proceedings of the Tenth Conference on Foundations of Software Technology and Theoretical Computer Science*, FST and TC 10, pages 316–331, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[AG94] A. Arora and M.G. Gouda. Distributed reset. *IEEE Trans. Computers*, 43(9) :1026–1038, 1994.

[AGBC11] N. Accettura, L.A. Grieco, G. Boggia, and P. Camarda. Performance Analysis of the RPL Routing Protocol. In *ICM*, 2011.

[AK93] S. Aggarwal and S. Kutten. *Foundations of Software Technology and Theoretical Computer Science : 13th Conference Bombay, India, December 15–17, 1993 Proceedings*, chapter Time optimal self-stabilizing spanning tree algorithms, pages 400–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[AKY91] Y. Afek, S. Kutten, and M. Yung. Memory-efficient self stabilizing protocols for general networks. In *Proceedings of the 4th International Workshop on Distributed Algorithms*, WDAG '90, pages 15–28, London, UK, UK, 1991. Springer-Verlag.

[APSVD94] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev. Self-stabilization by local checking and global reset (extended abstract). In *Proceedings of the 8th International Workshop on Distributed Algorithms*, WDAG '94, pages 326–339, London, UK, UK, 1994. Springer-Verlag.

[AS97] G. Antonoiu and P. Srimani. A self-stabilizing distributed algorithm to find the center of a tree graph. *Parallel Algorithms and Applications*, 10(3-4) :237–248, 1997.

[BBD15] L. Blin, F. Boubekeur, and S. Dubois. A self-stabilizing memory efficient algorithm for the minimum diameter spanning tree under an omnipotent daemon. In *2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 1065–1074, 2015.

[BBD16] L. Blin, F. Boubekeur, and S. Dubois. Algorithme auto-stabilisant efficace en mémoire pour la construction d'un arbre couvrant de diamètre minimum. In *18èmes Rencontres Francophones pour les Aspects Algorithmiques des Télécommunications, AlgoTel 2016, France, Bayonne, May 24-27, 2016*, 2016.

[BBK+15] L. Baron, F. Boubekeur, R. Klacza, M. Yasin Rahman, C. Scognamiglio, N. Kurose, T. Friedman, and S. Fdida. Demo : Onelab : Major computer networking testbeds for iot and wireless experimentation. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking, MobiCom 2015, Paris, France, September 7-11, 2015*, pages 199–200, 2015.

[BBLM15] F. Boubekeur, L. Blin, R. Léone, and P. Medagliani. Bounding degrees on RPL. In *Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Q2SWinet 2015, Cancun, Mexico, November 2-6, 2015*, pages 123–130, 2015.

[BDPBR10] L. Blin, S. Dolev, M. Potop-Butucaru, and S. Rovedakis. Fast self-stabilizing minimum spanning tree construction. In *DISC'10*, pages 480–494, 2010.

[BF15] L. Blin and P. Fraignaud. Space-Optimal Time-Efficient silent Self-Stabilizing constructions of constrained spanning trees. In *The 35th International Conference on Distributed Computing Systems (ICDCS 2015)*, pages 589–598, Columbus, USA, June 2015.

[BFP14] L. Blin, P. Fraigniaud, and B. Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *16th International Symposium Stabilization, Safety, and Security of Distributed Systems, (SSS 2014)*, pages 18–32, 2014.

[BGJ99] J. Beauquier, M. Gradinariu, and C. Johnen. Memory space requirements for self-stabilizing leader election protocols. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, PODC, '99Atlanta, Georgia, USA, May 3-6, 1999*, pages 199–207, 1999.

[BGKP99] S. Bruell, S. Ghosh, M. Karaata, and S. Pemmaraju. Self-stabilizing algorithms for finding centers and medians of trees. *SIAM Journal of Computing*, 29(2) :600–614, 1999.

[BK07] J. Burman and S. Kutten. Time optimal asynchronous self-stabilizing spanning tree. In *DISC'07*, pages 92–107, 2007.

[BLB95] F. Butelle, C. Lavault, and M. Bui. A uniform self-stabilizing minimum diameter tree algorithm (extended abstract). In *WDAG'95*, pages 257–272, 1995.

[Bli11] L. Blin. *HDR : Algorithmes auto-stabilisants pour la construction d'arbres couvrants et la gestion d'entités autonomes.* PhD thesis, Université Pierre et Marie Curie, 2011.

[BPBR11] L. Blin, M. Potop-Butucaru, and S. Rovedakis. Self-stabilizing minimum degree spanning tree within one from the optimal degree. *J. of Parallel and Distributed Computing*, 71(3) :438–449, 2011.

[BPR09a] L. Blin, M. Gradinariu Potop-Butucaru, and S. Rovedakis. Self-stabilizing minimum-degree spanning tree within one from the optimal degree. In *23rd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2009, Rome, Italy, May 23-29, 2009*, pages 1–11, 2009.

[BPR09b] L. Blin, M. Gradinariu Potop-Butucaru, and S. Rovedakis. A superstabilizing $\log(n)$-approximation algorithm for dynamic steiner trees. *CoRR*, abs/0902.3528, 2009.

[BPR13] L. Blin, M. Potop-Butucaru, and S. Rovedakis. A super-stabilizing log(n)log(n)-approximation algorithm for dynamic steiner trees. *Theor. Comput. Sci.*, 500 :90–112, 2013.

[BPRT09] L. Blin, M. Gradinariu Potop-Butucaru, S. Rovedakis, and S. Tixeuil. A new self-stabilizing minimum spanning tree construction with loop-free property. *CoRR*, abs/0905.2287, 2009.

[BPRT16] L. Blin, M. Gradinariu Potop-Butucaru, S. Rovedakis, and S. Tixeuil. Loop-free super-stabilizing spanning tree construction. *Comput. J.*, 59(3) :225–243, 2016.

[BT13] L. Blin and S. Tixeuil. Compact deterministic self-stabilizing leader election - the exponential advantage of being talkative. In *DISC'13*, pages 76–90, 2013.

[CABM03] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-throughput Path Metric for Multi-hop Wireless Routing. In *MobiCom*, pages 134–146, 2003.

[CD94] Z. Collin and S. Dolev. Self-stabilizing depth-first search. *Information Processing Letters*, 49(6) :297–301, 1994.

[CdVJY+12] T. Clausen, A. C. de Verdiere, A. Niktash J. Yi, Y. Igarashi, and U. Herberg. The Lightweight On-demand Ad hoc Distance-vector Routing Protocol - Next Generation (LOADng), October 2012.

[CHP11] T. H. Clausen, U. Herberg, and M. Philipp. A critical evaluation of the IPv6 Routing Protocol for Low Power and Lossy Networks (RPL). In *WiMob*, pages 365–372, 2011.

[CPMD15] D. Carels, E. Poorter, I. Moerman, and P. Demeester. Rpl mobility support for point-to-point traffic flows towards mobile nodes. *International Journal of Distributed Sensor Networks*, 2015, January 2015.

[CRV11] A. Cournier, S. Rovedakis, and V. Villain. The first fully polynomial stabilizing algorithm for BFS tree construction. In *Principles of Distributed Systems - 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings*, pages 159–174, 2011.

[DGS96] S. Dolev, M.G. Gouda, and M. Schneider. Memory requirements for silent stabilization (extended abstract). In *15th Annual ACM Symposium on Principles of Distributed Computing, (PODC 1996)*, pages 27–34. ACM, 1996.

[DGV04] A. Dunkels, B. Gronvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *LCN*, pages 455–462, 2004.

[Dij74] E. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communication of the ACM*, 17(11) :643–644, 1974.

[DIM90] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. In *(PODC'90*, pages 103–117, 1990.

[DIM93] S. Dolev, A. Israeli, and S. Moran. Self-stabilization of dynamic systems assuming only read/write atomicity. *Distributed Computing*, 7(1) :3–16, 1993.

[DJ15] S. Devismes and C. Johnen. Silent self-stabilizing BFS tree algorithms revised. *CoRR*, abs/1509.03815, 2015.

[DJPV00] A. K. Datta, C. Johnen, F. Petit, and V. Villain. Self-stabilizing depth-first token circulation in arbitrary rooted networks. *Distrib. Comput.*, 13(4) :207–218, November 2000.

[DL13] A. Datta and L. Larmore. Leader election and centers and medians in tree networks. In *SSS'13*, pages 113–132, 2013.

[DLV08] A. K. Datta, L. L. Larmore, and P. Vemula. Self-stabilizing leader election in optimal space. In *Stabilization, Safety, and Security of Distributed Systems, 10th International Symposium, SSS 2008, Detroit, MI, USA, November 21-23, 2008. Proceedings*, pages 109–123, 2008.

[DLV11] A. Datta, L. Larmore, and P. Vemula. An o(n)-time self-stabilizing leader election algorithm. *J. Parallel Distrib. Comput.*, 71(11) :1532–1544, 2011.

[Dol93] S. Dolev. *Distributed Algorithms : 7th International Workshop, WDAG'93 Lausanne, Switzerland, September 27–29, 1993 Proceedings*, chapter Optimal time self stabilization in dynamic systems, pages 160–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[Dol00] S. Dolev. *Self-stabilization.* MIT Press, March 2000.

[DT11] S. Dubois and S. Tixeuil. A taxonomy of daemons in self-stabilization. Technical Report 1110.0334, ArXiv eprint, October 2011.

[Dun11] A. Dunkels. The ContikiMAC Radio Duty Cycling Protocol, 2011.

[EMM11] B. Emmanuel, P. Matthias, and G. Mukul. The p2p-rpl routing protocol for ipv6 sensor networks : Testbed experiments. In *The P2P-RPL Routing Protocol for IPv6 Sensor Networks : Testbed Experiments*, pages 1 – 6, 2011.

[EÖF+09] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. Marrón. COOJA/MSPSim : interoperability testing for wireless sensor networks. In *ICSTT*, page 27, 2009.

[FMNA15] E. Fleury, N. Mitton, T. Noel, and C. Adjih. FIT IoT-LAB : The Largest IoT Open Experimental Testbed. *ERCIM News*, page 14, 2015.

[FR94] M. Fürer and B. Raghavachari. Approximating the Minimum-Degree Steiner Tree to within One of Optimal. *J. Algorithms*, 17(3) :409–423, 1994.

[GB81] E. M. Gafni and P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 29(3) :11–18, 1981.

[GBS00] S. Gupta, A. Bouabdallah, and P. Srimani. Self-stabilizing protocol for shortest path tree for multi-cast routing in mobile networks (research note). In *Euro-Par'00*, pages 600–604, 2000.

[GFJ+13] O. Gnawali, R. Fonseca, K. Jamieson, M. A. Kazandjieva, D. Moss, and P. Levis. CTP : an efficient, robust, and reliable collection tree protocol for wireless sensor networks. *TOSN*, 10(1) :16, 2013.

[GK12] O. Gaddour and A. Koubaa. RPL in a nutshell : A survey. *Computer Networks*, 56(14) :3163–3178, 2012.

[GKBA14] O. Gaddour, A. Koubaa, N. Baccour, and M. Abid. OF-FL : QoS-aware fuzzy logic objective function for the RPL routing protocol. In *WiOpt*, pages 365–372, 2014.

[GL12] O. Gnawali and P. Levis. The Minimum Rank with Hysteresis Objective Function. RFC 6719 (Standards Track), September 2012.

[GMFL14] P. Gonizzi, P. Medagliani, G. Ferrari, and J. Leguay. RAWMAC : A routing aware wave-based MAC protocol for WSNs. In *WiMob*, pages 205–212, 2014.

[GS99] S. K. S. Gupta and Pradip K. Srimani. Using self-stabilization to design adaptive multicast protocol for mobile ad hoc networks. *In Workshop on Mobile Networks and Computing*, 1999.

[GS03] S. K. S. Gupta and P. K. Srimani. Self-stabilizing multicast protocols for ad hoc networks. *J. Parallel Distrib. Comput.*, 63(1) :87–96, January 2003.

[HC92] S.-T. Huang and N.-S. Chen. A self-stabilizing algorithm for constructing breadth-first trees. *Information Processing Letters*, 41(2) :109–117, 1992.

[HC93] S.-T. Huang. and N.-S. Chen. Self-stabilizing depth-first token circulation on networks. *Distributed Computing*, 7(1) :61–66, 1993.

[HC11] U. Herberg and T.H. Clausen. A comparative performance study of the routing protocols LOAD and RPL with bi-directional traffic in low-power and lossy networks (LLN). In *Proceedings of the 8th ACM Symposium on Performance evaluation of wireless ad hoc, sensor, and ubiquitous networks,*

*PE-WASUN 2011, Miami Beach, Florida, USA, October 31 - November 4, 2011*, pages 73–80, 2011.

[HL01] L. Higham and Z. Liang. *Distributed Computing : 15th International Conference, DISC 2001 Lisbon, Portugal, October 3–5, 2001 Proceedings*, chapter Self-stabilizing Minimum Spanning Tree Construction on Message-Passing Networks, pages 194–208. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[HL02] T.C. Huang and Ji-C. Lin. A self-stabilizing algorithm for the shortest path problem in a distributed system. In *Computers and Mathematics with Applications*, page 103 – 109, 2002.

[HT95] R. Hassin and A. Tamir. On the minimum diameter spanning tree problem. *Information Processing Letters*, 53(2) :109–111, 1995.

[HT11] J. Hui and P. Thubert. Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. RFC 6282 (Proposed Standard), September 2011.

[Hua05] S.-T. Huang. A self-stabilizing algorithm for the shortest path problem assuming read/write atomicity. *J. Comput. Syst. Sci.*, 71(1) :70–85, 2005.

[HW97] S-T. Huang and L-C. Wuu. Self-stabilizing token circulation in uniform networks. *Distributed Computing*, 10(4) :181–187, 1997.

[HW12] S. Holzer and R. Wattenhofer. Optimal distributed all pairs shortest paths and applications. In *PODC'12*, pages 355–364, 2012.

[ITN13] O. Iova, F. Theoleyre, and T. Noël. Stability and efficiency of RPL under realistic conditions in Wireless Sensor Networks. In *PIMRC*, pages 2098–2102, 2013.

[JT03] C. Johnen and S. Tixeuil. Route preserving stabilization. In *Proceedings of the 6th International Conference on Self-stabilizing Systems*, SSS'03, pages 184–198, Berlin, Heidelberg, 2003. Springer-Verlag.

[KET+11] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-haggerty, A. Terzis, A. Dunkels, and D. Culler. Contikirpl and tinyrpl : Happy together. In *In Proceedings of the workshop on Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, 2011.

[KKM11] A. Korman, S. Kutten, and T. Masuzawa. Fast and compact self stabilizing verification, computation, and fault detection of an MST. In *PODC'11*, pages 311–320, 2011.

[KP93] S. Katz and K.J. Perry. Self-stabilizing extensions for message-passing systems. *Distrib. Comput.*, 7(1) :17–26, November 1993.

[KRS84] E. Korach, D. Rotem, and N. Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Transactions on Programming Languages and Systems*, 6(3) :380–401, 1984.

[KSS12] K. Dominik Korte, A. Sehgal, and J. Schönwälder. A study of the rpl repair process using contikirpl. In *AIMS*, pages 50–61, 2012.

[LCC+12] N. Thanh Long, N. Caro, W. Colitti, A. Touhafi, and K. Steenhaut. Comparative performance study of RPL in wireless sensor networks. In *19th IEEE Symposium on Communications and Vehicular Technology in the Benelux, SCVT 2012, Eindhoven, The Netherlands, November 16, 2012*, pages 1–6, 2012.

[LCH+11] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko. The Trickle Algorithm. RFC 6206 (Proposed Standard), March 2011.

[LHD13] CA. La, M. Heusse, and A. Duda. Link reversal and reactive routing in Low Power and Lossy Networks. In *PIMRC*, pages 3386–3390, 2013.

[LSD+12] K.C. Lee, R.S. Sudhaakar, L. L. Dai, S. Addepalli, and M. Gerla. RPL under mobility. In *2012 IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, January 14-17, 2012*, pages 300–304, 2012.

[MFAM13] P. Di Marco, C. Fischione, G. Athanasiou, and PV. Mekikis. MAC-aware routing metrics for low power and lossy networks. In *INFOCOM*, pages 13–14, 2013.

[MKHC07] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks, 2007.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, July 2003.

[Per85] Radia J. Perlman. An algorithm for distributed computation of a spanning tree in an extended LAN. In *SIGCOMM '85, Proceedings of the Ninth Symposium on Data Communications, British Columbia, Canada, September 10-12, 1985*, pages 44–53, 1985.

[PRT12] D. Peleg, L. Roditty, and E. Tal. Distributed algorithms for network diameter and girth. In *ICALP'12*, pages 660–672, 2012.

[PTD11] B. Pavkovic, F. Theoleyre, and A. Duda. Multipath opportunistic RPL routing over IEEE 802.15.4. In *MSWiM*, pages 179–186, 2011.

[PV99] F. Petit and V. Villain. Time and space optimality of distributed depth-first token circulation algorithms. In *WDAS'99*, pages 91–106, 1999.

[Rov09] S. Rovedakis. *Algorithmes auto-stabilisants de constructions d'arbres couvrants*. PhD thesis, Université d'Evry-Val-d'Essonne, 2009.

[RSA12a] I. E. Radoi, A. Shenoy, and D. K. Arvind. Evaluation of Routing Protocols for Internet-Enabled Wireless Sensor Networks, 2012.

[RSA12b] I.E. Radoi, A. Shenoy, and D. Arvind. Evaluation of routing protocols for internet-enabled wireless sensor networks, 2012.

[RW13] L. Roditty and V. Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *STOC'13*, pages 515–524, 2013.

[ST11] L. Ben Saad and B. Tourancheau. Sinks mobility strategy in ipv6-based wsns for network lifetime improvement. In *4th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2011, Paris, France, February 7-10, 2011*, pages 1–5, 2011.

[TdOV10a] J. Tripathi, J. Cavalcante de Oliveira, and J-P. Vasseur. A performance evaluation study of rpl : Routing protocol for low power and lossy networks. In *CISS*, pages 1–6, 2010.

[TdOV10b] J. Tripathi, J. Cavalcante de Oliveira, and JP. Vasseur. A performance evaluation study of RPL : Routing Protocol for Low power and Lossy Networks. In *CISS*, pages 1–6, 2010.

[Thu12] P. Thubert. Objective Function Zero for the Routing Protocol for Low-Power and Lossy Networks (RPL). RFC 6552 (Standards Track), March 2012.

[VKP+12] J.P. Vasseur, M. Kim, K. Pister, N. Dejean, and D.Barthel. Routing Metrics Used for Path Calculation in Low-Power and Lossy Networks. RFC 6551 (Standards Track), March 2012.

[VRG+14] M. Vucinic, G. Romaniello, L. Guelorget, B. Tourancheau, F. Rousseau, O. Alphand, A. Duda, and L. Damon. Topology construction in RPL networks over beacon-enabled 802.15.4. *CoRR*, 2014.

[VTD14] M. Vucinic, B. Tourancheau, and A. Duda. Performance comparison of the RPL and loadng routing protocols in a home automation scenario. *CoRR*, abs/1401.0997, 2014.

[WAA11] T. Watteyne, M. R. Akhavan, and A. H. Aghvami. Enhancing the performance of RPL using a receiver-based MAC protocol in lossy WSNs. In *ICT*, pages 191–194, 2011.

[WTB+12a] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL : IPv6 Routing Protocol for Low-Power and Lossy Networks, 2012.

[WTB+12b] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL : IPv6 Routing Protocol for Low-Power and Lossy Networks, March 2012.

[WTZ+10] D. Wang, Z. Tao, J. Zhang, A. Abouzeid, and A. alhussein. Rpl based routing for advanced metering infrastructure in smart grid. In *International Conference communications(ICC)*, May 2010.

# Table des figures


1.1 Réseau de capteurs . . . . . 8
1.2 Exemple d'arbre couvrant de diamètre minimum. . . . . 10
1.3 Exemple d'arbre couvrant de degré borné. . . . . 11

2.1 Algorithme silencieux et algorithme bavard . . . . . 20
2.2 Algorithmes auto-stabilisants asynchrones pour la construction d'arbres couvrants. . . . . 23
2.3 Exemple d'arbre couvrant de diamètre minimum. . . . . 27

3.1 Circulation du jeton . . . . . 36
3.2 Illustration de la fonction p_next. . . . . 40
3.3 Example de calcul de l'excentricité. . . . . 44
3.4 Ordre de priorité des règles. . . . . 48
3.5 Sous-structures couvrantes possibles en l'absence de construction du `Backbone`. . . 49
3.6 Application de la règle $\mathbb{R}_{\texttt{ErToken}}$ dans un sous-arbre $S_{TreeFR}$. . . . . 50
3.7 Application de la règle $\mathbb{R}_{\texttt{ErToken}}$ dans une structure couvrante $S_{Cycle}$. . . . . 50
3.8 Sous-structures couvrantes . . . . . 52
3.9 Sous-structures couvrantes. . . . . 54
3.10 Illustration de la variable p_next (flèches bleues) dans un sous-arbre couvrant avec des nœuds activables par des règles du `Backbone` ou $\mathbb{R}_{\texttt{ErToken}}$ . . 56

4.1 Capteur OpenMote-CC2538 . . . . . 66
4.2 Empilement de couches pour un réseau de capteurs 6LoWPAN. . . . . 68
4.3 Trame d'un paquet de contrôle ICMPv6 . . . . . 73
4.4 Message ICMPv6 . . . . . 74
4.5 DODAG comparé à un arbre couvrant . . . . . 75
4.6 Construction d'un DODAG selon la métrique ETX . . . . . 77

5.1 Nombre de changements de parent et nombre de paquets perdus en fonction du temps de simulation. . . . . 85
5.2 DODAG RPL comparé à un DODAG BD-RPL. . . . . 88

5.3 DODAG non mis à jour. . . . . 89
5.4 DODAG mis à jour. . . . . 90
5.5 Tableau résumant les éléments de la configuration choisis. . . . . 93
5.6 Consommation de l'énergie radio en fonction de la distance en nombre de sauts. . . . . 94
5.7 Délai de transmission ascendant en fonction de la distance en nombre de sauts. . . . . 95
5.8 Délai de transmission descendant en fonction de la distance en nombre de sauts. . . . . 95
5.9 Taux de délivrance de paquets ascendant en fonction de la distance en nombre de sauts. . . . . 96
5.10 Tableau résumant les choix de configuration pour l'expérimentation. . . . . 97
5.11 Nombre moyen de changements de parent en fonction de la distance en nombre de sauts. . . . . 97
5.12 Nombre de changements de parent en fonction du temps. . . . . 98
5.13 Consommation de l'énergie radio en fonction du nombre de sauts. . . . . 98
5.14 Délai de transmission en fonction du nombre de sauts. . . . . 99
5.15 Taux de délivrance de paquets en fonction du nombre de sauts. . . . . 99
5.16 Taux de délivrance de paquets en fonction du temps. . . . . 100

# Les arbres couvrants de la théorie à la pratique
## Algorithmes auto-stabilisants et Réseaux de Capteurs

**Résumé :** Les réseaux de capteurs sont des réseaux particuliers composés d'objets contraints en ressources. Ils possèdent une faible puissance de calcul, une faible puissance de transmission, une faible bande passante, une mémoire de stockage limitée ainsi qu'une batterie à durée de vie limitée.

Afin d'intégrer de tels réseaux dans l'internet des objects, de nouveaux protocoles ont été standardisés. Parmi ces protocoles, le protocole RPL (pour Routing Protocol for Low Power and Lossy Networks). Ce protocole est destiné a construire une topologie logique de routage appelée DODAG. Dans cette thèse, nous abordons l'aspect acheminement de données qui considère une topologie de routage arborescente. L'acheminement des données se fait donc de saut en saut d'un enfant à son parent (ou d'un parent à son enfant). Optimiser la construction du DODAG revient donc à construire un arbre couvrant selon une contrainte donnée. Un arbre couvrant est une structure communicante qui permet de maintenir un unique chemin entre toutes paires de noeuds tout en minimisant le nombre de liens de communication utilisés. De plus, nous considérons les contraintes des réseaux de capteurs telles qu'une batterie déchargée et la variabilité du lien radio comme des fautes transitoires. Ceci nous conduit par conséquent à construire une structure couvrante tolérante aux fautes transitoires. L'auto-stabilisation est une branche de l'algorithmique distribuée qui assure qu'à la suite d'une ou de plusieurs fautes transitoires, le système va retrouver de lui-même un comportement correcte au bout d'un temps fini. L'objectif de cette thèse est de proposer des algorithmes auto-stabilisants dédiés aux réseaux de capteurs. Les contributions de la thèse sont les suivantes :

1. Dans la première partie de la thèse, nous avons proposé un algorithme auto-stabilisant pour la construction d'un arbre couvrant de diamètre minimum. Cette construction est naturelle lorsque nous souhaitons minimiser le délai de communication entre une racine et les tout les autres noeuds du réseau. Notre algorithme possède plusieurs avantages. Tout d'abord, notre algorithme se limite à une occupation mémoire de $O(\log n)$ bits par noeud, ce qui réduit le résultat précédent d'un facteur $n$ tout en conservant un temps de convergence polynomial. De plus, notre algorithme est le premier algorithme pour la construction d'un arbre couvrant auto-stabilisant qui fonctionne sous un démon distribué non équitable. En d'autres termes, nous ne faisons aucune restriction sur le comportement asynchrone du réseau.
2. Dans la deuxième partie de la thèse, nous nous sommes intéressés à la topologie instable construite par le protocole RPL (DODAG). Notre solution consiste à placer une contrainte additionnelle sur le nombre d'enfants qu'un nœud peut accepter durant la construction du DODAG. Cette contrainte a pour effet de réduire le taux de changement de parent et par conséquent d'améliorer les performances du protocole en termes de taux de délivrance de paquets, de délai de communication et de consommation énergétique. De plus, nous avons implémenté un mécanisme afin de mettre à jour les informations sur les routes descendantes dans RPL. Notre solution a aussi l'avantage de ne pas générer un surplus de messages de contrôle car nous utilisons les messages de contrôle existants fournis par RPL pour l'implémenter. Enfin,

cette contribution est double puisque nous avons validé notre solution à la fois en simulations et en expérimentations.

**Mots clés :** Auto-stabilisation, Arbres couvrants, Réseaux de Capteurs, Évaluation des performances, Protocole RPL

# Spanning Trees from theory to practice
## Self-stabilizing Algorithms and Sensor Networks

**Abstract :** Sensor networks are composed of ressources constrained equipments. They have low computing power, low transmission power, low bandwidth, limited storage memory and limited battery life.

In order to integrate such networks in the Internet of things, new protocols were standardized such as RPL protocol (for Routing Protocol for Low Power and Lossy Networks). This protocol is intended to build a logical routing topology called DODAG (for Destination Oriented Directed Acyclic Graph). In this thesis, we discuss the data routing aspect by considering a tree routing topology. Thus, the routing of data is hop by hop from a child to its parent (or from a parent to its child). Optimize the construction of the DODAG is therefore to build a spanning tree in a given constraint. A spanning tree is a connecting structure that maintains a unique path between all pairs of nodes while minimizing the number of used communication links. Furthermore, we consider the constraints of sensor networks, such as a dead battery and the variability of the radio link as transient faults. This leads us to build a covering structure tolerant to transient faults. The self-stabilization is a branch of distributed algorithms that ensures that following one or more transient faults, the system will find itself a correct behavior after a finite time. The objective of this thesis is to propose self-stabilizing algorithms dedicated to sensor networks. The contributions of this thesis are :

1. The first part of the thesis is theoretical, we proposed a self-stabilizing algorithm for the construction of a minimum diameter spanning tree. This construction is natural when we want to minimize the communication delay between a root and all other network nodes. Our algorithm has several advantages. First, our algorithm is limited to memory occupation of $O(\log n)$ bits per node, reducing the previous result of an $n$ factor while maintaining a polynomial convergence time. Then, our algorithm is the first algorithm for minimum diameter spanning tree that works as an unfair distribution demon. In other words, we make no restriction on the asynchronous network behavior.
2. In the second part of the thesis, we are interested in the unstable topology built by RPL protocol (DODAG). Our solution is to place an additional constraint on the number of children a node can accept during the construction of the DODAG. This constraint has the effect of reducing the rate of parent change and consequently to improve the protocol performance in terms of packet delivery rate, delay of communication and power consumption. In addition, we implemented a mechanism to update the information of the downward routes in RPL. Furthermore, our solution has the advantage of not generating overhead because we use existing control messages provided by RPL to implement it. Finally, this contribution is twofold since we validated our solution both by simulations and experiments.

**Keywords :** Self-stabilizing, Spanning Trees, Sensor Networks, Performance Evaluation, RPL Protocol