**UNIVERSITÉ CLERMONT AUVERGNE**

ÉCOLE DOCTORALE
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

**THÈSE**

Présentée par

**MOUSSA DETHIÉ SARR**

pour obtenir le grade de

**DOCTEUR D'UNIVERSITÉ**

SPÉCIALITÉ : INFORMATIQUE

# Spécification d'un mécanisme de construction automatique de topologies et d'adressage permettant la gestion dynamique des réseaux de capteurs sans fil linéaires

Soutenue publiquement le 17 Janvier 2018 devant le jury :

| | | |
|---|---|---|
| Rapporteurs : | Mme. KARA Naidja | Professeur à l'Université du Quebec |
| | Mme. MINET Pascale | Chargée de recherche HDR à INRIA Paris |
| | M. PHAM Congduc | Professeur à l'Université de Pau et des Pays de l'Adour |
| Directeur : | M. MISSON Michel | Professeur à l'Université Clermont Auvergne |
| Co-Encadrants : | M. NIANG Ibrahima | Maitre de Conférences, HDR à l'Université Cheikh Anta Diop |
| | M. DELOBEL François | Maitre de Conférences à l'Université Clermont Auvergne |
| Invités : | Mme. SERVAJEAN Marie-Françoise | Maitre de Conférences à l'Université Clermont Auvergne |
| | M. BOSSCHE Adrien Van Den | Maitre de Conférences à l'Université de Toulouse II Jean Jaures |

# Résumé

Les réseaux de capteurs sans fils linéaires (RdCSFL) sont un cas particulier de réseaux de capteurs sans fils où les nœuds de capteurs sont déployés le long de multiples lignes. les RdCSFL sont utilisés pour la surveillance des infrastructures routières, ferroviaires, des conduites de gaz, d'eau, de pétrole et de cours d'eau. Les solutions classiques de formation de topologie et d'adressage proposées ne sont pas adaptées à l'environnement des RdCSFL. En effet les paramètres initiaux utilisés par ces protocoles tels que le nombre maximum de nœuds fils ($Cm$), nombre maximum de nœuds routeurs fils $Rm$, profondeur maximum de l'arbre ($Lm$), occasionnent un gaspillage de l'espace d'adressage disponible pour les nœuds et limitent la profondeur de l'arbre adressable ($15$ sauts pour ZigBee). D'autres solutions adaptées pour les RdCSFL utilisent une organisation en cluster des nœuds du réseau et sont basées elles aussi sur des paramètres fixés à l'avance tels quel le nombre maximum de cluster fils par cluster. De plus, ces solutions requièrent beaucoup d'interventions manuelles sur les nœuds de capteurs (choix des chefs de cluster par exemple) et ne favorisent pas une adaptation face aux changements du RdCSFL tels que l'ajout d'un ensemble de nœuds de capteurs.

Dans cette thèse, nous proposons donc des protocoles permettant la construction automatique de topologies logiques, l'adressage et le routage pour des réseaux de capteurs sans fil linéaires. Nos protocoles fournissent aussi des mécanismes de gestion dynamique d'un RdCSFL avec l'ajout de nouveaux nœuds, la réallocation d'adresses pour les nœuds en cas d'épuisement de blocs d'adresses et la gestion du routage vers plusieurs puits du réseau. Nos différents protocoles sont évalués grâce au simulateur Castalia/Omnet++. Les résultats de nos simulations montrent que nos protocoles permettent de construire un RdCSFL connecté avec peu de nœuds orphelins (nœuds sans adresses logiques) et sans limitations de profondeur. Nous montrons aussi, grâce à nos simulations, que nos contributions permettent d'ajouter un grand nombres de nœuds à un RdCSFL existant de n'importe quelle taille et s'adaptent au déploiement de plusieurs puits et au routage multi-puits et permettent d'améliorer le ratio et la latence de paquets livrés dans les RdCSFL.

**Mots-clés** : Réseaux de capteurs sans fil linéaires, construction de topologies, déploiement automatique, adressage, routage, multi-puits

# Abstract

Linear wireless sensor network (LWSN) are a sub-case of wireless sensor network where sensor nodes are roughly deployed through multiple long lines with branches. LWSN are used to monitor infrastructures such as roads, pipelines, and naturals entities such as rivers. Classical solutions of topology construction and addressing are inefficient on LWSN . Indeed, with initials networks parameters such as the maximum number of children per node ($Cm$), the maximum number of children routers per node ($Rm$), and the maximum tree depth, a solution like ZigBee causes a waste of available address space of network nodes and limit the depth of the addressable tree to $15$ hops. Other solutions proposed for LWSN use a cluster-tree organisation and are based on initial network parameters such as the maximum number of children clusters per cluster. In addition, these solutions require a lot of manual intervention on different sensor nodes and do not allow adaptation for a network extension (addition of a set of new sensor nodes).

In this thesis, we propose protocols to allow the automatic construction of topologies, the addressing and the data routing for linear wireless sensor networks. Our contribution also provides mechanisms for dynamic management of LWSN (addition of new nodes, addresses reallocation, and data routing to multiple sink nodes). Our different protocols are evaluated using Castalia/Omnet++ simulator. Results of our simulations show that our protocols allow a construction of connected LWSN with very few orphan nodes and without depth limitations. We also show that our contribution allows to add many new nodes on different LWSN, and adapts to the deployment of multiple sinks to improve the ratio and the latency of data delivery packets.

**Keywords** : Linear wireless sensor network, topology construction, addressiing, routing, multi-sink

# Remerciements

Cette thèse n'aurait jamais abouti aux présents résultats sans l'aide et le soutien de plusieurs institutions et de nombreuses personnes.

Je tiens à exprimer ma plus grande gratitude à mes directeurs de thèse M. Michel Misson, M. Ibrahima Niang et M. François Delobel qui m'ont fait confiance et m'ont donné l'opportunité d'effectuer ce travail de thèse riche et très formateur au sein de l'équipe réseaux et protocoles de l'AXE SIC du LIMOS.

À M. Congduc Pham, professeur à l'université de Pau, Mme Kara Naidja, professeur à l'université du Quebec et à Mme Pacale Minet, chargée de recherche-HDR à l'INRIA qui ont accepté de rapporter mon manuscrit de thèse. Je souhaite les adresser mes chaleureux remerciements pour le temps passé, pour leur enthousiasme, et pour leur lecture attentive du manuscrit et leurs précieux conseils qui m'ont permis d'améliorer ce travail. Je remercie également Mme Marie-Françoise Servajean, maître de conférences et M. Adrien Van Den Bossche, maître de conférences pour l'intérêt qu'ils ont porté à mon travail en me faisant l'honneur de faire partie de mon jury.

Je profite de cette occasion pour témoigner toute ma reconnaissance au Professeur Matar Mour Seck, Recteur de l'Université de Thiès et au Professeur Cheikh Sarr, Directeur de l'UFR SET qui ont ménagé mes charges d'enseignements afin de me faciliter mes séjours au niveau de l'Université Clermont Auvergne ainsi qu'aux collégues du Département d'informatique et aux personnels de l'université de Thiès. Mes remerciements vont aussi aux Professeur Moussa Lô et Professeur Ousmane Thiaré et à toute l'équipe du CEA-MITIC (Centre d'excellence Africain) pour leur accompagnement.

Je tiens aussi à remercier l'école doctorale des sciences pour l'ingénieur (SPI), et en particulier à son directeur M. Therry Chateau pour ses conseils et sa disponibilité pour les démarches administratives. Je remercie également le directeur du laboratoire LIMOS, M. Farouk Toumani, son infatigable responsable administratif Mme. Béatrice Bourgier et tout le personnel administratif et enseignant du laboratoire pour leur disponibilité face aux demandes les plus diverses.

Un grand merci à tous les enseignants, doctorants et stagiaires de l'équipe réseaux et protocoles, qui ont toujours répondu présent pour répondre à mes questions et pour leur conseil et m'orienter dans ma recherche. Un grand merci à Alexandre Guitton, Frédérique Jacquet, Gérard Chaloub, Antonio Frétas, Joël Toussaint pour ces conseils et son soutien durant ces années. Merci aussi à Thérèse, Honoré, Xavier, David, Malick, Hamadoun et Sidibé, Guenne, Mouna et toutes les personnes que j'ai eu à côtoyer au niveau du bureau C6 pour les échanges très constructifs et leur soutien.

Je tiens aussi à remercier tout le personnel du département d'informatique et réseau et télécom de l'IUT de Clermont. Toute ma reconnaissance va aussi à l'égard de mes amis de Clermont pour leur encouragement, leur soutien et avec qui j'ai passé d'excellents moments de partage durant ces années. Une attention particulière va aussi à l'égard Valériane Précope.

Une attention particulière à mon encadreur M. François Delobel pour tout son soutien sans faille qui m'a permis de passer des séjours agréables riches en enseignements, en compétences techniques du monde linux et en découvertes de la région d'Auvergne.

Je ne saurais terminer sans adresser tout ma reconnaissance et à tous mes amis du Sénégal pour le soutien et leur encouragement mais surtout toute ma gratitude à toute ma famille pour leur soutien et leurs prières.

# Table des matières

# Table des figures

# Liste des tableaux

# Introduction

Les Réseaux de Capteurs Sans Fil (RdCSF) sont une collection de nœuds de capteurs sans fil qui collaborent, collectent et transmettent des données. Ces RdCSF peuvent être utilisés dans des environnements [Aky+02] très variés tels que les volcans [Lor+06], les cours d'eau [Pan+08], les conduites de gaz et pétrole [Jaw+07], les lignes de chemins de fer [Zim+08], ponts [Kim+07], la santé [Bak+07]). Les avantages des réseaux de capteurs sans fil par rapport à ces types d'environnements reposent sur :

— leur facilité de déploiement : le déploiement d'un RdCSF ne nécessite pas une infrastructure réseau pré-existante. En, effet les RdCSF sont souvent utilisés pour la surveillance de milieux difficilement accessibles et sans aucune infrastructure réseau (zone volcanique, zone maritime, forêt, rivière et cours d'eau),

— leur grande capacité d'auto-organisation : après leur déploiement et leur mise en service, les nœuds de capteurs ont le plus souvent la capacité d'effectuer une organisation automatique en étoile, en arbre, ou maillée et le plus souvent sans aucune intervention (*manuelle*) de la part d'une tierce personne. En effet, en fonction de la zone à surveiller, les nœuds capteurs peuvent être déployés de façon aléatoire par largage à partir d'avions, uniforme sous forme de grille pour quadriller une zone sensible, ou linéaire le long d'une frontière ou d'un cours d'eau,

— et leur autonomie d'énergie : la plupart des nœuds de capteurs peuvent fonctionner durant des mois voir des années sans remplacement de leur batterie.

Cependant, les nœuds de capteurs sans-fils présentent quelques contraintes que sont une source d'énergie limitée (batteries ou piles), une capacité limitée en mémoire et en traitement des données, un faible débit [Aky+02 ;

Yic+08]. Ces contraintes font que la plupart des travaux de recherche dans le domaine des RdCSF visent à proposer des protocoles et mécanismes allant dans le sens de l'optimisation de la gestion de ces ressources.

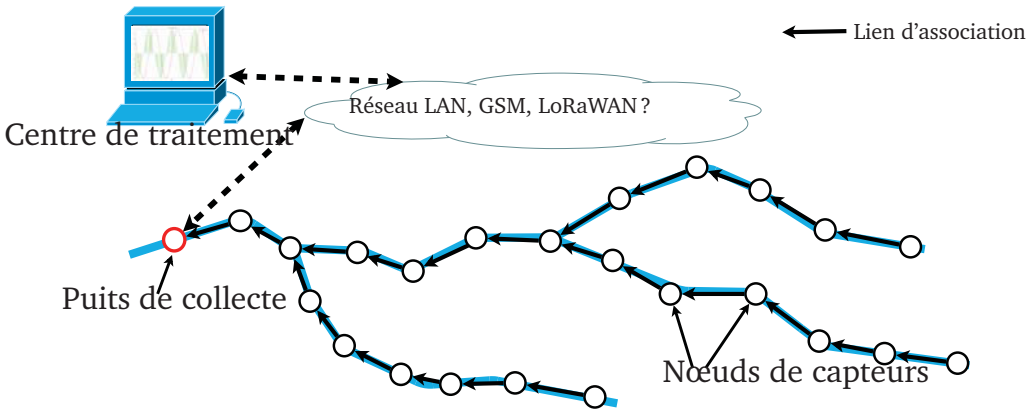



**Fig. 0.1.:** Réseau de capteurs sans fil pour la surveillance d'un cours d'eau

Le RdCSF de la figure 0.1 est composé de nœuds de capteurs jouant le rôle de capteur de données et de relais et d'équipements appelés puits permettant d'organiser le réseau et de recevoir et de traiter les données transmises par les nœuds. La figure 0.2 montre un exemple d'architecture de nœud de capteurs composé d'un module d'acquisition, d'une unité de traitement, d'une unité de communication et d'un module de gestion d'énergie. L'unité d'acquisition permet de connecter un capteur ou plusieurs capteurs pour la mesure des données telles que la température, l'humidité, la luminosité, etc. avec l'unité de traitement. Pour le cas de capteurs analogiques, un convertisseur analogique numérique ou CAN est nécessaire afin de convertir les données analogiques mesurées en données numériques avant de les transmettre à l'unité de traitement. L'unité de traitement est composée d'un module de stockage des données avant ou après leur traitement par le microprocesseur. L'unité de communication quant à lui comprend un module radio pour l'émission des données mesurées à destination d'autres nœuds de capteurs et la réception des données venant d'autres nœuds. L'unité de gestion d'énergie permet de fournir, aux différents composants du nœud de capteurs, l'énergie nécessaire pour son fonctionnement grâce à des batteries ou piles le plus souvent. Cependant d'autres sources d'énergie peuvent être utilisées comme les énergies renouvelables [AR15] afin d'améliorer l'autonomie en énergie des nœuds de capteurs.

Dans les RdCSF, l'environnement de déploiement détermine fortement la topologie ainsi que la taille du réseau. Dans un déploiement à l'intérieur

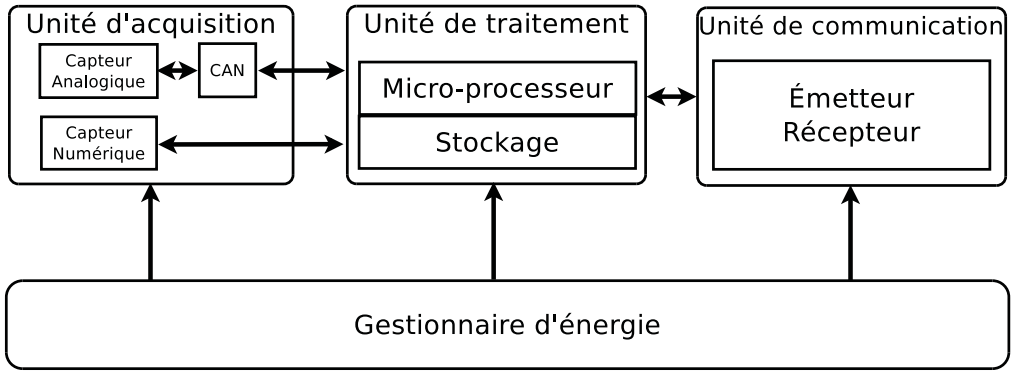



**Fig. 0.2.:** Architecture d'un nœud de capteurs sans-fil

d'un bâtiment ou d'un environnement fermé tel qu'un hôpital, un terrain de stade, une voiture, le RdCSF est composé de quelques dizaines de nœuds alors que dans le cas d'un environnement ouvert (océan, forêt, cours d'eau, volcan), des centaines de nœuds seront nécessaire pour couvrir la zone à surveiller.

Il existe aussi des environnements et infrastructures tels que les cours d'eau, les ponts, les autoroutes, les canalisations où la disposition des nœuds de capteurs forme des portions linéaires pouvant être reliés entre eux [Pan+08 ; Jaw+11 ; Zim+08 ; Kim+07]. Ces types d'environnements linéaires ont donné naissance à de nouveaux types de topologies de réseaux de capteurs appelés *réseaux de capteurs sans fil linéaires* (RdCSFL) qui sont des cas particuliers de topologies en arbre. Pour le déploiement sur ces environnements linéaires, chaque nœud de capteurs du RdCSFL a le plus souvent au moins deux voisins (un en direction du puits et un autre vers la direction contraire) dépendant de la portée du signal radio sauf pour les nœuds d'extrémité qui ont un seul voisin.

Dans [Jaw+11], les auteurs définissent un RdCSFL comme étant un réseau où les nœuds de capteurs sont déployés le long de segments linéaires ou semi-linéaires. Les nœuds de capteurs d'un tel réseau peuvent être alignés en ligne droite ou placés uniformément entre deux lignes parallèles.

Dans [Ndo15], les auteurs proposent une définition d'un RdCSFL composé d'un segment linéaire en fonction du facteur de redondance $K$. Ce paramètre $K$ représente le nombre de nœud voisins d'un nœud dans chaque direction. De ce fait, un RdCSFL $K$-redondant représente un réseau où chaque

nœud est à portée de transmission de $K$ voisins dans la direction du puits et autant de voisins dans la direction contraire.

Cependant, les RdCSFL sont généralement composés de plusieurs branches linéaires reliées à des endroits appelés zones de branchement [Pan+08] (voir figure 0.1). Dans ce cas, par rapport au type de déploiement, il existe des nœuds d'un tel RdCSFL qui auront un nombre de voisins supérieur à $2K$ nœuds.

Dans ce manuscrit de thèse, nous considérons qu'un réseau de capteurs est linéaire si une majorité de ses nœuds a au plus $2K$ voisins. Plus précisément par "majorité des nœuds", nous considérons qu'un RdCSF peut être plus ou moins linéaire et introduisons dans la partie 2.1.3 à la page 73 la notion de *facteur de linéarité* qui correspond au pourcentage de nœuds avec au plus $2K$ voisins.

Dans les RdCSFL, les paquets générés par les nœuds sont transmis de proche en proche jusqu'au nœud puits en passant par les mêmes chemins et sur de longues distances (voir Figure 0.1). Cette contrainte des réseaux linéaires introduit un important délai de bout en bout et peu de fiabilité par rapport aux réseaux sans fils classiques. De plus, dans la plupart des RdCSFL, chaque nœud joue le rôle de capteurs de données et aussi de nœud relais pour le routage multi-saut en direction du nœud puits. Ce qui fait que le débit des paquets drainés, à partir des nœuds sources lointains jusqu'au puits, augmente rapidement à chaque saut. Ce phénomène engendre des problèmes de congestion et de débordement de files d'attente au niveau des nœuds proches du puits mais aussi un nombre important de pertes de paquets dû aux collisions et aux échecs de transmissions de paquets.

Ces caractéristiques des RdCSFL (nombre de voisins limité, chemins de routage limité pour atteindre le puits , nombre de sauts très grand pour atteindre le puits) montrent qu'il est important de fournir une organisation de la topologie logique et un schéma d'adressage capable d'offrir un réseau pleinement connecté pour les collectes et le drainage des données de chaque nœud. La plupart des mécanismes d'auto-organisation et d'auto-adressage proposés pour les RdCSF tels que les mécanismes d'adressage distribué pour les topologies en arbre ZigBee [All08 ; KJ11 ; Par+09] ne sont pas adaptés pour de grands réseaux comme les RdCSFL. En effet, ces mécanismes ne

définissent pas la manière dont les nœuds doivent s'associer pour former les différentes topologies. Pour une topologie en arbre, lorsque les nœuds de capteurs sont disposés autour d'un nœud déjà connecté, il est difficile de prédire à l'avance si les nœuds vont devenir tous des fils de ce nœud ou bien former une chaîne linéaire (en consommant la profondeur). Ceci, d'une part, rend difficile le dimensionnement des paramètres initiaux tels que le nombre de fils maximum par nœud et la profondeur maximum afin d'éviter le problème d'apparition de nœuds orphelins ou d'inégales répartitions des blocs d'adresses [All08] et d'autre part peut générer des tables de routage complexes [KJ11] ou des mécanismes de gestions de conflits et/ou beaucoup de messages de contrôle de routage [Par+09].

Des solutions de mécanismes d'auto-organisation et d'adressage sont proposés pour les RdCSFL mais manquent de flexibilité lors des phases d'initialisation et de déploiement des nœuds mais aussi d'extension d'un RdCSFL. Dans [Pan+08] et dans [Jaw+11], les auteurs proposent des mécanismes d'organisation en cluster et d'adressage des RdCSFL. Cependant, ces propositions souffrent de beaucoup d'interventions manuelles tant pour le choix des paramètres initiaux des clusters qu'à l'assignation des adresses pour les nœuds. Par exemple [Pan+08], le dimensionnement du réseau en différents clusters et le calcul de l'identifiant de chaque cluster basé sur l'approche d'adressage des arbres de ZigBee se font manuellement avant le déploiement des nœuds.

Face à ces manquements, l'objectif de cette thèse est de proposer un mécanisme d'auto-organisation et d'auto-adressage spécialement adapté à la structure en arbre particulière d'un RdCSFL. Cet objectif a pour but le routage des informations capturées par les nœuds de capteurs tout en minimisant les interventions manuelles de l'administrateur du réseau. Ce mécanisme doit être dynamique pour permettre non seulement l'extension du réseau avec l'ajout de nouveaux nœuds pour la surveillance de nouvelles parties d'une autoroute par exemple mais aussi pour s'adapter à la défaillance d'un ou de plusieurs liens causés par la disparition de nœuds causée par l'épuisement de leur énergie ou par un élément extérieur bloquant le signal des nœuds. Nos différentes contributions portent sur les couches routage et application et permettront à chaque nœud du RdCSFL d'obtenir une adresse logique par rapport à sa position dans le réseau et de router les données collectées grâce à la seule connaissance de l'adresse de destination sans

utiliser de messages de contrôle de routage supplémentaires. Grâce à ces fonctionnalités qui permet d'ajouter de nouveaux nœuds tels que des puits secondaires 0.3, nous proposerons aussi un autre mécanisme de routage permettant le routage des données à destinations des puits en fonction de leur position (nombre de sauts du nœud source).

Ce travail de thèse a été entrepris dans le cadre des activités traitant des réseaux de capteurs sans fil du LIMOS (Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes)/CNRS.

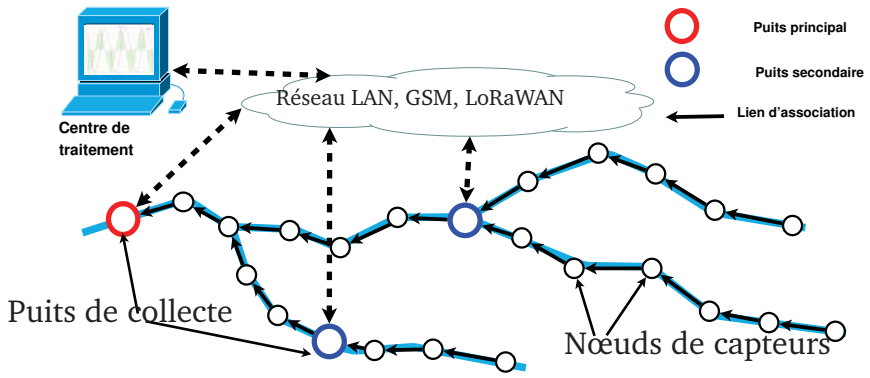



**Fig. 0.3.:** Réseau de capteurs sans fil comportant plusieurs puits de collecte pour la surveillance d'un cours d'eau

Ce manuscrit est organisé comme suit. Dans le chapitre 1, nous présenterons l'état de l'art des topologies de RdCSF, des mécanismes de contrôle de topologie dans les RdCSF, des protocoles MAC dans les RdCSF linéaires, des protocoles d'adressage et de routage dans les RdCSF et linéaires, et des mécanismes de routage multi-puits. Nous montrerons les inconvénients de ces mécanismes lorsque nous souhaitons les utiliser dans les RdCSF linéaire. Dans le chapitre 2, nous détaillerons notre première contribution sur le protocole de découverte, d'auto-adressage et de routage dans un RdCSF linéaire. Nous commencerons par présenter une nouvelle définition des RdCSFL avec l'introduction d'une nouvelle métrique appelée *facteur de linéarité* (lF) permettant de mesurer la linéarité d'un RdCSF. Ensuite, nous décrirons les différentes approches centralisée et distribuée de notre mécanisme avant de finir par l'évaluation des résultats de nos simulations. Le chapitre 3 abordera notre deuxième contribution sur le protocole de gestion de l'adressage et du routage dans un RdCSF linéaire dynamique où il est possible d'ajouter de nouveaux nœuds éparses ou bien des branches de nœuds. Dans cette partie, nous présenterons les différentes fonctionnalités de notre protocole pour l'ajout de nouveaux nœuds et la gestion de la réallocation de nouveaux

blocs d'adresses en cas d'épuisement sur un nœud, avant de finir par une évaluation de cette contribution via des résultats de simulations. Pour toutes nos contributions, les simulations réalisées avec le simulateur Castalia/Omnet++ [Ath] seront présentées. Enfin dans le chapitre 4, nous conclurons ce travail et présenterons les différentes perspectives pour la suite à donner à nos travaux.

# 1 État de l'art

Dans les réseaux de capteurs sans fil, l'objectif principal des nœuds de capteurs est la collecte et l'acheminement des données vers un centre de traitement. Pour ce faire, les nœuds déployés s'auto-organisent pour former un réseau de communication grâce à leur interface sans-fil afin de relayer les données collectées (éventuellement de proche en proche) jusqu'au centre de traitement en passant par une ou plusieurs passerelles du réseau de capteurs appelées puits (figures 0.1 et 0.3).

Dans les environnements linéaires longs (ligne de frontières, chemin de fer, autoroute, cours d'eau), différents protocoles sont proposés, en prenant en compte la spécificité du milieu, d'une part afin d'optimiser les ressources limitées des nœuds de capteurs du réseau et d'autre part pour permettre une auto-organisation rapide et un acheminement efficace des données collectées au centre de traitement.

Dans ce chapitre, nous présentons quelques applications et domaines d'utilisations des RdCSF. Ensuite, nous présentons les principaux standards pour les réseaux de capteurs sans fil pour lesquels nous aurons à décrire les caractéristiques et fonctionnement de quelques protocoles MAC les plus utilisés. Puis, nous aborderons les mécanismes de contrôle de topologie existant dans les RdCSFL utilisés pour l'organisation des nœuds de capteurs. Nous continuons par la présentation des différents types de topologies de RdCSFL et nous terminons par les mécanismes de routages multi-puits dans les RdCSF.

## 1.1 Applications des réseaux de capteurs sans fil

Les RdCSF peuvent être utilisés dans différents secteurs [Yic+08 ; RR16] pour faciliter la surveillance d'un milieu (température, humidité, détecteur de mouvement), de personnes (rythme cardiaque, glycémie, tension ) ou le

suivi et la localisation des objets et des animaux. Dans la suite, nous présentons quelques exemples de champs d'application des RdCSF.

Dans le cas des applications de surveillance, les RdCSF sont généralement utilisés pour mesurer, collecter et traiter les données afin de prendre des décisions au niveau du centre de surveillance ou de réaliser des actions grâce à d'autres types de nœuds appelés *actionneurs*.

### 1.1.1 Applications dans le domaine de la santé

Dans le domaine de la santé, les RdCSF peuvent être déployés dans les services d'urgence de l'hôpital pour la surveillance en temps réels des données vitales des patients. Les nœuds de capteurs sont alors utilisés pour mesurer et collecter les données telles que la tension, la température, le rythme cardiaque. des patients et les transmettre aux équipements de surveillance et aux écrans placés au niveau des salles de garde des personnels de santé [Bak+07 ; Gao+08 ; Jeo+10]. Pendant un sinistre ou une catastrophe, les RdCSF permettent à grand échelle le suivi, l'inventaire, la classification et la prise en charges des blessés par niveau d'urgence [Jeo+09].

Ce même dispositif de RdCSF peut être utilisé pour la surveillance des personnes âgées et des patients à leur domicile [Bos+16 ; CL13]. Dans ce cas, les nœuds de capteurs sont déployés dans les maisons et/ou porter par des personnes et les données vitales mesurées sont transmises à leur médecin ou infirmière grâce à d'autres systèmes de communication tels que IP, cellulaire (3G ou 4G).

### 1.1.2 Applications dans le domaine environnemental

Les RdCSF peuvent aussi être déployés dans une ville pour mesurer les niveaux de pollution causés par les usines et les voitures et décider par exemple d'appliquer la circulation alternée [Ma+08 ; Jun+11 ; Mao+12]. Dans [Jun+11], les auteurs proposent une infrastructure composée de nœuds de capteurs sans fil appelés nœuds *GeoSensor* qui permettent de mesurer la pollution, les radiations ultra-violets, la température, etc. Ces nœuds sont

déployés sur les zones à surveiller et les données mesurées sont transmises à un serveur qui se charge de faire l'analyse des données et l'émission de messages d'alertes sur certaines zones à risque à destination de la population. Dans [Ma+08], les auteurs proposent un autre type d'architecture pour la ville de Londres pour la surveillance du niveau de pollution causée par les émissions de gaz des véhicules. L'architecture proposée est composée de nœuds de capteurs fixes appelés *collecteurs*, placés le long des routes, et de nœuds de capteurs mobiles placés sur les véhicules (bus, taxis, véhicules de service ou commerciaux) qui mesurent les niveaux de pollution. Ces mesures sont par la suite transmises au nœud de capteurs fixe le plus proche qui se charge de faire la collecte et l'agrégation des données avant de les envoyer au serveur de traitement et de surveillance.

Dans les régions montagneuses, afin de réagir très rapidement face à des risques de chutes de débris sur les routes ou d'avalanches durant les périodes de skis, les RdCSF peuvent être mis à contribution [AM08]. Dans la prévention des catastrophes naturelles de grandes ampleurs telles que les tsunamis , les éruptions volcaniques, les feux de forêts, les RdCSF permettent d'alerter la population et les services d'urgence et de secours [Cas+08 ; Lor+06]. Dans le cas par exemple de la prévention des tsunamis, les auteurs de [Cas+08] proposent un déploiement de nœuds de capteurs dans l'océan afin de mesurer les variations de la pression de l'eau qui est le plus souvent un signe annonciateur d'un tsunamis et de déclencher les alarmes des villes côtières

### 1.1.3 Applications dans le suivi des objets, animaux ou personnes

Dans le domaine d'application des RdCSF pour le suivi et la localisation des objets, des animaux et des personnes, nous pouvons citer le domaine militaire où les nœuds de capteurs placés sur les véhicules et les personnes permettent une meilleure coordination et un déploiement plus efficace des troupes sur le terrain. Les RdCSF peuvent aussi être utilisés pour le suivi d'animaux, de véhicules ou de colis afin de pouvoir les localiser en temps réel dans de vastes régions comme les parcs nationaux, les domaines classés ou protégés, les villes [Jua+02 ; Sik+06 ; GB08 ; Gut+09].

### 1.1.4 Applications dans le domaine domotique

Dans le domaine de l'habitat avec l'émergence des objets connectés et des maisons intelligents, nous assistons à une très forte croissance de nouvelles applications des RdCSF destinées aux maisons. Dans [GP10], les auteurs décrivent un exemple d'architecture composé de nœuds de capteurs déployés dans une maison et permettant de mesurer la luminosité, la température, l'humidité, la présence de fumée ou de gaz. Suivant les données collecter en temps réels, il est alors décidé, grâce à des actionneurs connectés au RdCSF, d'ouvrir ou de fermer les stores, d'allumer ou d'éteindre la climatisation ou le chauffage ou bien d'activer les alarmes incendies.

### 1.1.5 Applications dans les infrastructures linéaires

Pour la surveillance des infrastructures telles que les ponts, les gratte-ciel, les tunnels, des nœuds de capteurs sont utilisés afin de mesurer les vibrations, inclinaisons, distorsions, etc. et anticiper tout risque d'effondrement [Sha+07 ; Fra+09 ; Kim+07 ; Zar+11]. Dans [See+11], les auteurs proposent un mécanisme permettant la surveillance des réseaux d'égouts contre tout risque d'obstruction et de déversement des eaux usées sur la voie publique. Cette infrastructure est composée de nœuds de capteurs placés le longs des égouts pour mesurer les différents niveaux de l'eau et les transmettre au centre de surveillance.

Dans le secteur de la distribution de gaz, de pétrole, d'eau, etc., des fuites dans les canalisations peuvent occasionner des pertes considérables en chiffre d'affaires mais aussi des problèmes d'ordre écologique ou sanitaire. Pour améliorer la surveillance d'un telle infrastructure et les interventions des services de maintenance, les RdCSF peuvent être déployés pour mesurer la pression et le débit circulant dans ces canalisations afin de détecter des fuites et d'alerter les services de maintenance ou d'actionner si nécessaire des vannes de fermeture pour éviter des dommages énormes [Jaw+07 ; Iva+07 ; Wan+11 ; Yoo+11].

Dans le secteur de la distribution d'électricité, les auteurs de [Lim+10] proposent un système de surveillance des pylônes et des postes de transformation haute tension grâce à des nœuds de capteurs. Ces nœuds de capteurs permettent de détecter les problèmes de surtension lors du transport de l'électricité et de surchauffe des postes de transformation.

## 1.2 Les standards dans les réseaux de capteurs sans fil

Les réseaux de capteurs sans-fil sont des réseaux composés de nœuds ayant des modules radio caractérisés par des débits de transmission très limités et pouvant fonctionner à de faible puissance d'émission. Suivant la technologie radio utilisée, la portée de transmission de ces nœuds peut couvrir de très petites distances (50 à 100 mètres) ou bien atteindre de très grandes distances (jusqu'à 50 kilomètres).

En fonction des caractéristiques telles que la puissance de transmission, l'énergie consommée et le débit de transmission des données, les protocoles et technologies proposés sont regroupés en deux grandes familles. La famille des "Low-Rate Wireless Networks" [Lrw] qui regroupe les protocoles, standards et technologies pour les RdCSF fonctionnant à faible débit de transmission et sur de courte portée. La famille des "Low-Power Wide Area Network" [FE17] regroupe ceux fonctionnant à faible consommation énergétique, sur de longue portée et à faible débit de transmission. Dans cette partie, nous détaillerons quelques protocoles les plus utilisés dans ces deux grandes familles.

### 1.2.1 Low-Rate Wireless Personal Area Networks (LR-WPAN)

Les Low-Rate Wireless Personal Area Networks (LR-WPAN) concernent les RdCSF à faible coût de communication fournissant une connectivité sans-fil aux applications ayant des débits d'émission limitée. Les groupes de travail IEEE (Institute of Electrical and Electronics Engineer) et ZigBee Alliance ont

adopté des standards et normes à savoir la norme 802.15.4 [Iee ; Soc11] et la norme ZigBee [All08] (Voir figure 1.1). Le standard 802.15.4 définit les spécifications et mécanismes de la couche Physique et de la couche d'accès au médium ou MAC (Medium Access Control) pour les LR-WPAN. La norme ZigBee [All08] a pour but de définir les profils applicatifs et une spécification pour la couche réseau en s'appuyant sur la norme 802.15.4. Plusieurs autres protocoles proposés dans le domaine des réseaux de capteurs sans-fil pour les couches physiques, MAC et réseau s'appuient sur ces deux standards.

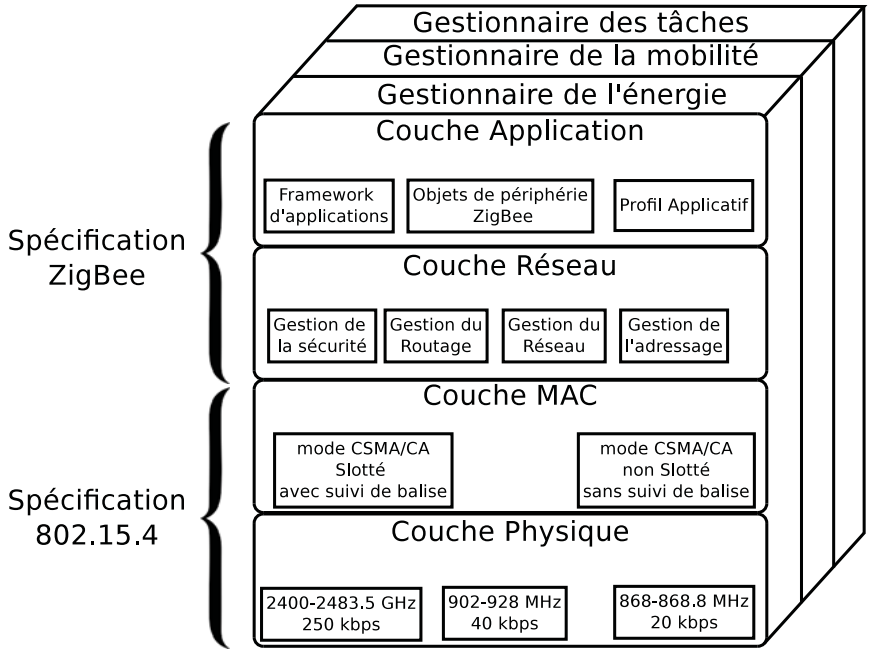



**Fig. 1.1.:** Pile protocolaire 802.15.4/ZigBee

### Le standard 802.15.4

La norme 802.15.4 [Iee ; Soc11] définit les spécifications au niveau physique et MAC que les nœuds doivent implémenter afin de répondre aux caractéristiques et exigences des réseaux de capteurs sans-fil se revendiquant de cette norme. Les spécifications au niveau de la couche physique définissent trois bandes de fréquences pour un total de 27 canaux :

— La bande $868.0 - 868.8$ MHz avec un canal à 20 Kbps ;

— La bande $902.0 - 928.0$ MHz avec 10 canaux à 40 Kbps ;

— La bande $2400 - 2483.5$ MHz avec 16 canaux à 250 Kbps.

Pour assurer la bonne transmission des données, la couche physique implémente les fonctionnalités de gestion du module radio (transmission, réception, veille), de sélection du canal, de gestion de la qualité du lien ou LQI (Link Quality indicator) et de gestion du canal (sélection du canal, test d'occupation du canal ou CCA (Clear Channel Assessment)) :

Un réseau personnel 802.15.4 ou PAN (Personal Area Network) est composé de nœuds de capteurs et d'un coordinateur du réseau appelé PAN coordinateur qui est un nœud assurant l'initialisation et la gestion du réseau. Le PAN coordinateur est le contrôleur principal d'un RdCSF. Les spécifications de la couche MAC 802.15.4 définissent deux types de nœuds de capteurs :

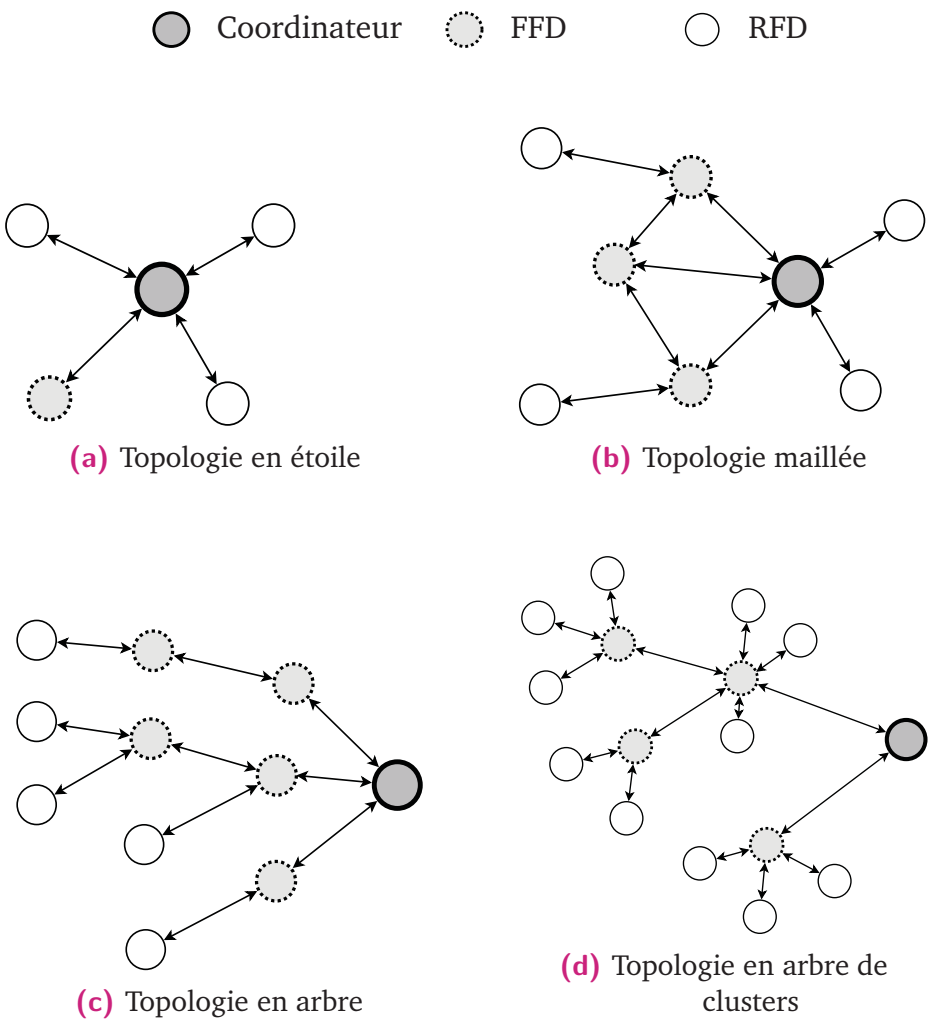
— Un nœud, avec toutes les fonctionnalités implémentées, appelé *nœud FFD* (*Full Function Device*). Ce type de nœud est utilisé pour jouer le rôle de coordinateur du réseau et de nœuds capables de relayer les données à destination du coordinateur ;

— Un nœud, avec des fonctionnalités limitées, appelé *nœud RFD* (*Reduced Function Device*). Ces types de nœuds sont le plus souvent déployés à la périphérie d'un réseau de capteurs sans-fil et sont utilisés pour la mesure des données de la zone de captage.

Dans les RdCSF, deux types de topologies sont définis : la topologie en étoile et la topologie maillée. La figure 1.2 montre une représentation de ces différentes topologies. Dans la topologie en étoile (voir Figure 1.2a), tous les nœuds du réseau envoient leurs données au coordinateur PAN. Ce type de topologie suppose que les nœuds du réseau soient à portée du coordinateur du réseau.

La topologie maillée ou pair à pair (voir Figure 1.2b) est utilisée dans le cas de grands réseaux où le coordinateur principal n'est pas à portée de transmission de tous les nœuds. Dans ce cas, les données collectées sont relayées par les nœuds intermédiaires jusqu'au coordinateur du réseau et un protocole de routage est le plus souvent nécessaire pour l'acheminement des données jusqu'au coordinateur PAN.

Il existe un cas particulier de topologie maillée appelé *topologie en arbre de clusters* (voir figure 1.2d) où les nœuds suivent une organisation hiérarchique en arbre. Chaque nœud envoie les données collectées à son nœud parent appelé chef de cluster qui se charge de son acheminement jusqu'au




**(a)** Topologie en étoile

**(b)** Topologie maillée

**(c)** Topologie en arbre

**(d)** Topologie en arbre de clusters

**Fig. 1.2.:** Types de topologies 802.15.4

puits. Dans ce type de réseau d'arbre de clusters, les données collectées d'un nœud transitent par un seul et unique chemin jusqu'au nœud coordinateur du réseau.

La couche MAC 802.15.4 implémente des fonctionnalités d'accès au support (couche MAC) basées soit sur l'approche sans contention, soit sur l'approche avec contention avec l'utilisation de l'algorithme d'accès multiple à détection de porteuse avec évitement de collision ou CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). 802.15.4 supporte deux modes de fonctionnement :

— Le mode suivi de balises où les balises sont diffusées périodiquement dans le réseau par le PAN coordinateur pour la synchronisation des nœuds. L'accès au médium est réalisé avec la méthode CSMA/CA slotté qui est une variante de CSMA/CA exploitant une synchronisation complète des nœuds ;

— Le mode sans suivi de balises. Avec ce mode de fonctionnement, l'accès au médium se base sur le mécanisme CSMA/CA non slotté qui est une méthode d'accès où il n'existe pas de synchronisation des nœuds du réseau. Chaque nœud peut transmettre ces données dès lors qu'il détecte que le canal est libre.

La balise permet de transmettre aux nœuds du réseau des informations de synchronisation ou *supertrame* qui définissent les périodes d'activité et de sommeil. Le format de la supertrame est défini par le PAN coordinateur et transmis à des intervalles de temps appelés BI (Beacon Interval). La durée de la période active de la supertrame est représentée par la valeur SD (Supertrame Duration). Ces deux valeurs sont calculées par le PAN coordinateur à partir des paramètres BO (macBeaconOrder) et SO (macSuperframeOrder) avec $0 \leq SO \leq BO \leq 14$. Les équations 1.1 et 1.2 montrent les fonctions de calcul de BI et SD en fonction de BO et SO avec $aBaseSuperframeDuration = 15.36\ ms$

$$BI = aBaseSuperframeDuration \times 2^{BO} \tag{1.1}$$

$$SD = aBaseSuperframeDuration \times 2^{SO} \tag{1.2}$$

La période active de la supertrame est divisée en $16$ slots de temps égaux qui sont répartis en trois parties : la balise, la période d'accès par contention CAP (Contention Access Period) et la période d'accès sans contention CFP

(Contention Free Period). Durant la période CAP, l'accès au médium se fait en utilisant le CSMA/CA slotté. Pour la période CFP, des slots de temps GTS (Guaranted Time Slot) sont réservés à chaque station voulant transmettre. De ce fait, l'accès au médium se fait sans l'utilisation du CSMA/CA slotté (sans contention).

La figure 1.3 montre un exemple d'une supertrame avec une période active comprenant une CAP et une CFP. La CFP, quant à elle, est répartie en deux GTS

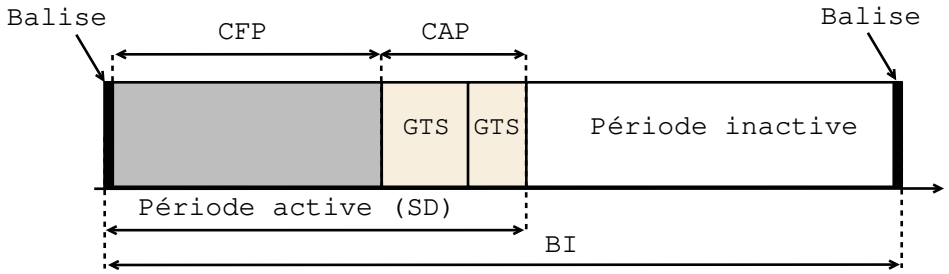



**Fig. 1.3.:** La structure de la supertrame de 802.15.4

La norme 802.15.4 définit uniquement les spécifications de la couche physique et de la couche MAC. D'autres protocoles tels que ZigBee proposent les spécifications et mécanismes des couches supérieures (réseau et application).

### Le standard ZigBee

[All08] et [Cun07] introduisent la norme ZigBee qui fournit les spécifications nécessaires pour les couches supérieures (réseau et application) en s'appuyant sur le standard IEEE 802.15.4 [Iee ; Soc11].

La couche réseau du standard ZigBee implémente les fonctionnalités nécessaires à la formation d'un réseau de capteurs, à l'adressage des nœuds, au routage des paquets à destinations des nœuds du réseau et en particulier du nœud puits (voir Figure 1.1 à la page 1.1). ZigBee définit trois types de rôles pour les nœuds de capteurs :

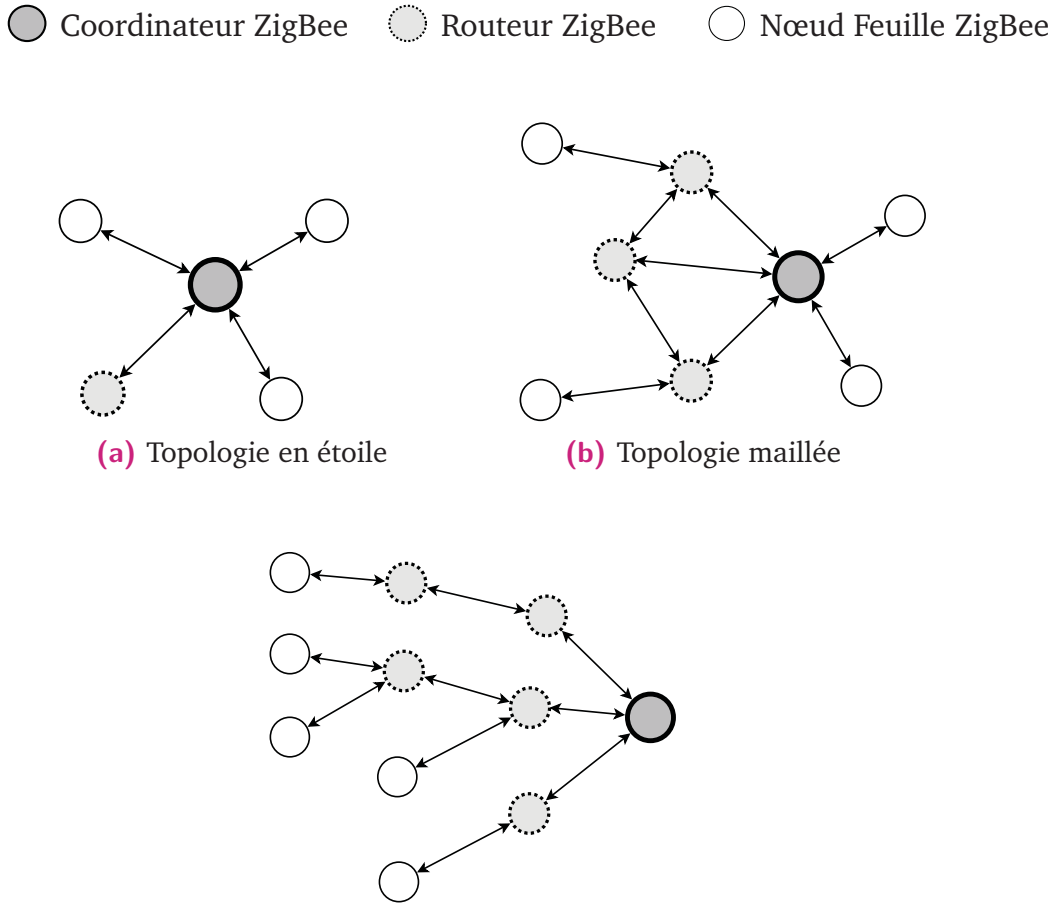
— Le **coordinateur ZigBee** qui correspond au nœud coordinateur appelé aussi coordinateur PAN ou puits. Ce type de nœud est un nœud FFD

(Full Function Device) de 802.15.4. Il est responsable de la création du réseau, de l'initialisation et de la gestion distribuée des adresses.

— Les **routeurs ZigBee** sont des nœuds FFD participant aux fonctions de routage multi sauts des données mais aussi faisant de la collecte de données car ils peuvent être équipés de capteurs.

— Les **nœuds feuilles ZigBee** sont des nœuds de type RFD (Reduce Function Device) c'est à dire des nœuds avec des fonctionnalités réduites et sont utilisés uniquement pour faire la capture des données qui seront transmises à leur nœud père.

La couche réseau de ZigBee est responsable de la formation du réseau, de ses évolutions (l'entrée et la sortie d'un nœud) et de l'attribution des adresses logiques aux nœuds.




**(a)** Topologie en étoile

**(b)** Topologie maillée

**(c)** Topologie en arbre

**Fig. 1.4.:** Types de topologies ZigBee

Le standard ZigBee offre deux mécanismes d'adressage logique. Le mécanisme d'adressage stochastique qui permet d'attribuer les adresses aléatoirement aux nœuds du réseau avec un mécanisme de détection et de sup-

pression des doublons. Ce mécanisme est le plus souvent utilisé dans les topologies en étoile (figure 1.4a) et pair à pair (figure 1.4b). Le mécanisme d'adressage distribuée, adapté quant à lui pour les réseaux d'arbre de clusters (figure 1.4c), permet d'assigner des adresses uniques et hiérarchiques aux nœuds du réseau.

### 1.2.2 Low Power Wide Area Network (LPWAN)

Low-Power WAN (LPWAN) [FE17 ; Raz+17] définit les RdCSF de type *machine-à-machine* (M2M) fonctionnant à très faible consommation énergétique et transmettant leurs données à très faible débit sur de longues portées. Les LPWAN ont introduit des nouveaux types d'applications de réseaux de capteurs couramment appelés *Internet des objets*. Le faible coût de ces réseaux LPWAN et le grand nombre de nœuds supportés sur de grandes zones de couverture ont favorisé l'émergence et la standardisation de nombreux protocoles et technologies tels que LoRaWAN [Sor+16], SIGFOX [Sig], NB-IoT (NarrowBand-Internet of Things) [Nbi], 6LPWAN, 802.15.4g, etc. Dans la suite, nous présenterons les technologies Sigfox et LoRaWan.

#### Sigfox

Sigfox [Sig ; Was] est un réseau d'un opérateur cellulaire propriétaire permettant de connecter des nœuds capteurs aux applications et services stockés sur des serveurs réseau grâce à un déploiement de stations de base ou passerelles. Sigfox utilise la technologie radio bande ultra étroite (100Hz) ou UNB (Ultra Narrow Band) et la modulation BPSK (Binary Phase Shift Keying) pour la communication entre les nœuds de capteurs et les stations de base. Grâce à cette technologie, quelques dizaines de hertz sont nécessaires pour la modulation et la transmission du signal sur de très longues portées ( jusqu'à 50 Km) avec une très faible consommation énergétique du module radio et à très bas débit (environ 100 bps).

Sigfox utilise les bande de fréquences ISM (868 MHz en Europe, 915 MHz aux USA). Les liens de communication entre les nœuds de capteurs (puces Sigfox) et les infrastructures (antennes radio) supportent le mode bidirectionnel. En Europe, Sigfox divise la bande de fréquences 868,18 à 868,220

MHz en 400 canaux de 100 Hz dont 40 sont réservés et non utilisés [Was]. Chaque station de base scanne chacun des 360 canaux afin de détecter et décoder toutes les transmissions. Les nœuds de capteurs choisissent aléatoirement un canal pour la transmission de leurs données vers la passerelle. Par défaut, chaque message est transmis trois fois vers les stations de base afin d'améliorer leur probabilité de réception.

### Long Range Wide Area Network (LoRaWAN)

LoRaWAN (Long Range Wide Area Network) [Sor+16] est une technologie sans-fil développée par l'alliance LoRa pour la communication à faible débit (entre $0.3$ kbps et $50$ kbps), faible puissance et longue portée (jusqu'à 50 Km). LoRaWAN définit les spécifications de la couche MAC pour la couche physique LoRa. LoRaWAN fait partie de la famille des technologies LPWAN qui couvre les applications d'internet des objets avec des caractéristiques telles qu'une large zone de couverture, une faible bande passante, une longue durée de vie des batteries. L'architecture de LoRaWAN est principalement basée sur une topologie en étoile composée de périphériques finaux (nœuds de capteurs, actionneurs, objets connectés), d'un serveur réseau et de passerelles jouant le rôle de pont transparent pour le relayage des messages entre les périphériques finaux et le serveur. La connexion entre les périphériques finaux et les passerelles est basée sur la communication à un saut LoRa (Long Range) ou FSK (Frequency Shift Keying) alors qu'elle est basée sur la communication IP entre les passerelles et le serveur. La figure 1.5 montre un exemple d'un réseau LoraWAN.

LoRa est une technologie de modulation à étalement de spectre de type *chirp* (Chirp Spread Spectrum) permettant des communications longues portées à faible puissance ce qui permet avec quelques de passerelles de couvrir une ville entière. Pour cela, Lora introduit la notion de facteur d'étalement (SF ou Spreading Factor) qui représente la longueur de code transmise. Avec un grand facteur d'étalement, le nœud a un rapport signal sur bruit (SNR) faible ce qui augmente sa sensibilité au récepteur de ce fait sa portée avec la gateway.

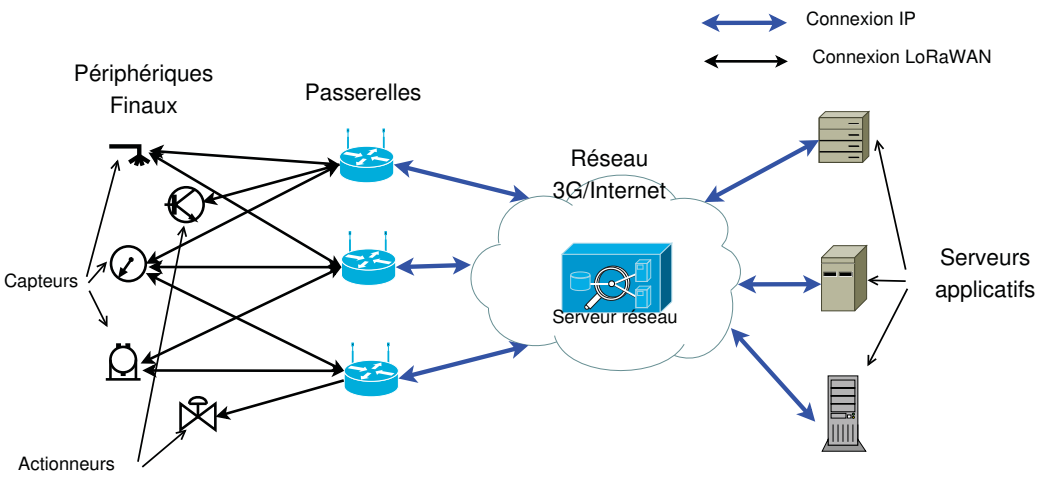



**Fig. 1.5.:** Architecture d'un réseau LoRaWAN

le tableau montre les différents facteur d'étalement supportés par Lora ainsi que leur débit. Les spécifications LoRaWAN définissent des bandes fréquences par zone géographique (Europe, États-unis, Asie, Australies, etc.).

— Europe : la bande $863.0 - 870.0$ MHz avec $8$ canaux de $125$ kHz permettant d'atteindre des débits de $0.25$ à $5.5$ kbps, un canal haut débit de 11 Kbps et un canal FSK de 50 Kbps ;

— États-unis : La bande $902.0 - 928.0$ MHz avec $64$ canaux de $125$ kHz et $8$ canaux de $500$ kHz pour les débits montants (des nœuds vers la passerelle), et $8$ canaux de $500$ kHz alloués aux débits descendants (de la paserelle vers les nœuds) ;

— Australie : La bande $915 - 928$ MHz a un fonctionnement similaire à celle la bande ISM des états-unis hormis le fait que les fréquences des liens montants se trouvent sur des fréquences beaucoup plus hautes que ceux des états-unis.

La figure 1.6 montre l'agencement des couches protocolaires LoraWan sur un nœud de capteurs.

LoRaWAN définit trois types de périphériques finaux tous permettant des communications bidirectionnelles et répartis en 3 classes A, B et C.

— Classe A - périphériques finaux bidirectionnels : Ils permettent une communication bidirectionnelle pour laquelle chaque transmission mon-

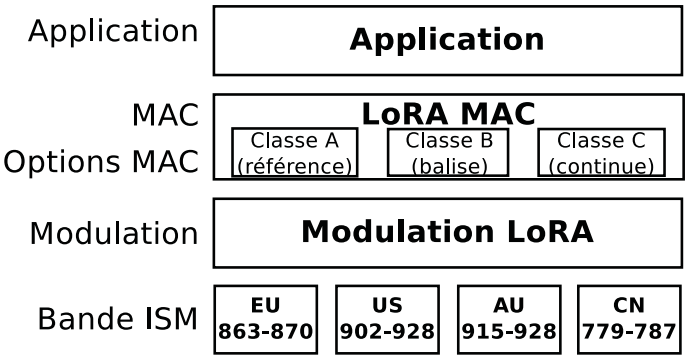



**Fig. 1.6.:** Pile protocolaire de LoraWAN sur un nœud

tante, déclenchée par le périphérique finale, est suivie de deux fenêtres de réception appelée *RX1* et *RX2*. C'est la classe de base supportée par tous les périphériques finaux et qui garantit une très grande économie d'énergie. La première fenêtre de réception est ouverte $1\ seconde \pm 20\ \mu S$ après la fenêtre de transmission et la deuxième fenêtre de réception démarre $1\ seconde \pm 20\ \mu S$ après le début de la première fenêtre de réception (voir figure 1.7a) ;

— Classe B - périphériques finaux bidirectionnels avec plage de réception programmée : En plus des deux fenêtres de réception de la classe A, la classe B ajoute des fenêtres de réception supplémentaires appelées *ping* sous forme de rendez-vous grâce aux balises envoyées périodiquement tous les $128\ s$ par la passerelle. La balise fournit les informations de programmation des fenêtres de réception durant cette période. La première fenêtre est définie aléatoirement et les autres fenêtres tous les $32\ s$. Durant les fenêtres de réception, les périphériques finaux vérifient la disponibilité de données à télécharger à partir de la passerelle. Afin d'éviter la fin prématurée d'un téléchargement de données par la réception d'une nouvelle balise durant la fenêtre de réception, la classe B définit un temps de *garde de la balise* de $3\ s$ durant lequel aucune fenêtre de réception ne peut être placée. Une période de $2.12\ s$ appelée *Réserve de la balise* suit automatique la période de garde. La période de réserve de la balise permet de protéger et de favoriser une bonne réception de la balise par les périphériques finaux. La figure 1.7b montre les différentes plages de temps pour la synchronisation d'un nœud de classe B ;

— Classe C - périphériques finaux bidirectionnels avec des plages de réception maximales : Ces types d'équipements n'ont pas de contrainte

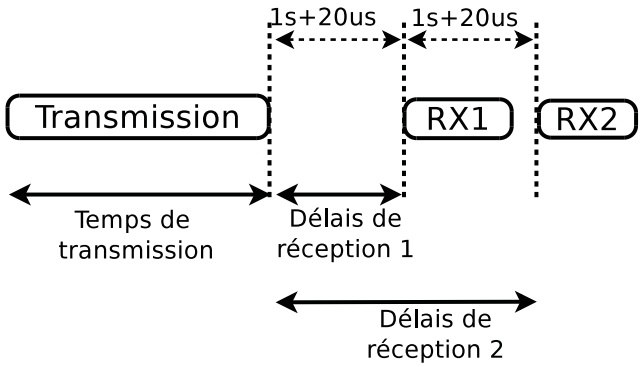
d'énergie et ont des fenêtres de réception ouvertes en continue et fermées qu'en période de transmission. La classe C implémente les mêmes deux plages de réception que la Classe A et permet l'écoute sur la fenêtre de réception RX2 tant qu'il n'est pas le moment d'ouvrir la fenêtre de réception RX1. De ce fait, pour pouvoir commencer à recevoir les données juste après la fin de la période de transmission et avant le début de la plage de réception RX1, la classe C autorise l'ouverture d'une courte fenêtre de réception sur RX2 (voir figure 1.7c).

Dans un réseau LoRaWAN, les messages émis par les périphériques finaux sont reçus par toutes les passerelles déployées dans la zone de couverture et transférés au serveur réseau de destination qui se charge de supprimer les données redondantes.

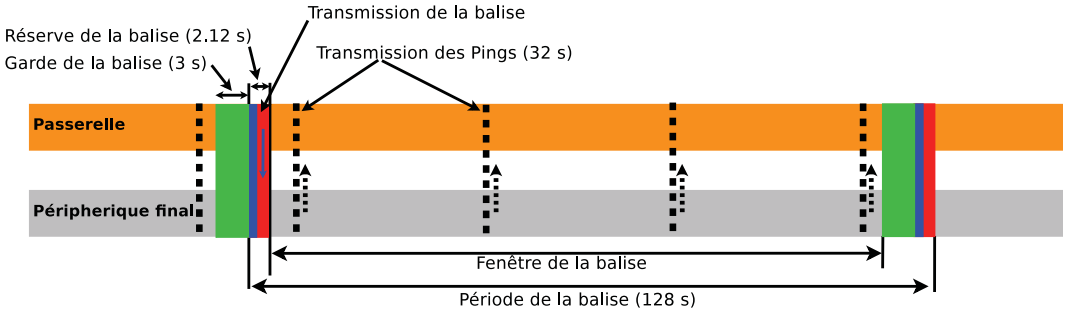
## 1.3 Le contrôle de la topologie dans les RdCSF

L'une des caractéristiques d'un réseau de capteurs sans-fil est sa capacité d'auto-organisation des nœuds afin de générer une certaine forme de topologie capable d'assurer une disponibilité du réseau pour l'acheminement des données capturées jusqu'au centre de collecte. Ces RdCSF sont le plus souvent aléatoirement et densément déployés. Les topologies résultantes sont dynamiques durant la durée de vie du réseau causées par la défaillance d'un lien, par la disparition d'un nœud ou par le changement de position d'un nœud. Ceci peut occasionner des partitionnements du réseau. Avec des ressources limitées en énergie et en portée de communication, il est important que cette topologie générée puisse fournir les meilleurs liens de communication entre les différents nœuds afin d'assurer les tâches d'acheminement des données collectées jusqu'aux nœuds puits ou centre de traitement tout en prolongeant la durée de vie du réseau.

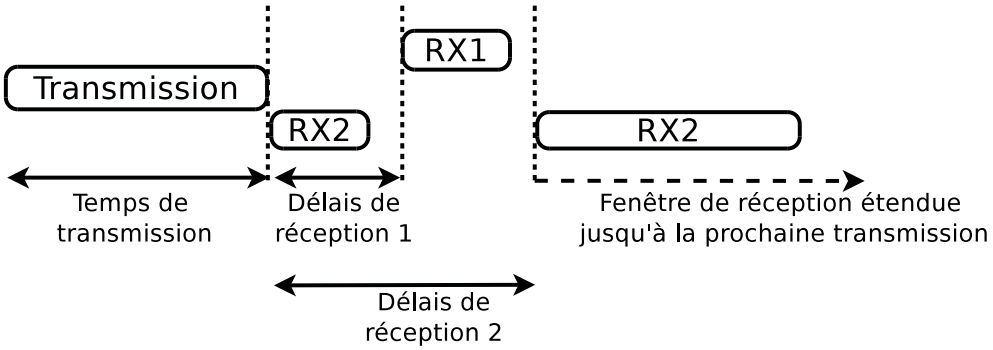
Les mécanismes de contrôle de la topologie dans les RdCSF permettent de générer et de maintenir la topologie d'un RdCSF. Il s'agit de décider à chaque instant de la formation de la topologie, quel nœud dans un ensemble donné sera autorisé à joindre le réseau en fonction de différentes métriques telles




**(a)** Classe A, point de vue d'une nœud final




**(b)** Classe B, répartition des slots de temps




**(c)** Classe C

**Fig. 1.7.:** Répartition des slots de temps LoRaWAN

que la proximité du nœud (ajustement de la puissance du signal afin de privilégier les courtes distances), l'énergie résiduelle, ou le nombre de voisins [San05 ; Yua+06a ; Tan+12 ; Azi+13]. Ces mécanismes offrent ainsi une utilisation plus efficace de la bande passante du réseau, tout en diminuant les interférences, et les retransmissions de paquets. Ceci permet une augmentation de la durée de vie des nœuds du réseau.

En fonction de ces métriques utilisées pour la formation des topologies de RdCSF, ces techniques de contrôle de la topologie peuvent être classées en quatre catégories : basées sur l'ajustement de la puissance d'émission, basées sur l'état du nœud (actif, veille), basées sur les clusters, basée un fonctionnement hybride combinant certains des techniques précédemment citées. Nous détaillerons ces mécanismes dans les sections suivantes.

### 1.3.1 Mécanismes basés sur l'ajustement de la puissance d'émission

Avec cette technique, les nœuds du réseau auront la possibilité de collaborer afin d'ajuster la puissance d'émission pour former un réseau connecté.

Les auteurs de [VTH99 ; LJY01] ont proposé des mécanismes appelés respectivement MECN (Minimum Energy Communication Network) et SMECN (Small Minimum Energy Communication Network) qui permettent de construire des topologies basées sur des liens consommant le moins d'énergie possible durant la transmission et le relayage des données vers le puits de collecte. Ces deux mécanismes nécessitent de pouvoir connaître la position géographiques des nœuds. Dans le mécanisme MECN [VTH99 ; Azi+13], les auteurs définissent un nœud voisin comme étant un nœud avec qui on peut échanger directement mais pour lequel il n'existe aucun autre nœud relais dans son voisinage qui permet de joindre ce nœud. Sur la figure 1.8, le nœud A peut directement communiquer avec tous les nœuds. Cependant il peut joindre aussi le nœud $C$ en passant par le nœud $B$. De ce fait, il ne va pas considérer $C$ comme voisin durant la phase de découverte de voisinage. Avec cette notion, chaque nœud effectue une phase de découverte de voisinage. Puis, les nœuds construisent une topologie (en graphe) avec les liens vers le puits consommant le minimum d'énergie grâce à l'algorithme

de Bellman-Ford. Chaque nœud diffuse le coût des liens avec chacun de ces nœuds voisins pour joindre le puits afin de permettre le routage des données.

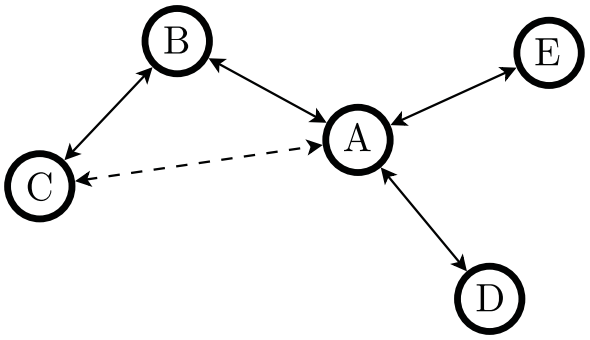



**Fig. 1.8.:** Voisinage du nœud A

Le coût du lien $Cout(a, b)$ entre un nœud $a$ et son voisin $b$ est calculé grâce à l'équation (1.3).

$$Cout(a, b) = Cout(b) + d(a, b)^n + \beta \tag{1.3}$$

$d(a, b)$ est la distance euclidienne entre les nœuds $a$ et $b$ en supposant que les nœuds sont équipés de GPS et ont la capacité de connaître leur position. $n$ représente l'exposant de la perte de chemin et $\beta$ est la puissance consommée durant la réception du message venant du nœud $b$. Pendant toute la durée de vie du réseau, MECN reconfigure si nécessaire la topologie en fonction des changements (ajout ou disparition d'un nœud, modification sur le chemin de propagation).

Le mécanisme SMECN [LJY01] propose une extension à MECN en permettant de construire un graphe plus petit que celui de MECN. Pour ce faire, SMECN considère qu'un nœud ajouté comme voisin ne peut être supprimé. Ceci fait que la topologie construite ne varie pas avec SMECN.

### 1.3.2 Mécanismes basés sur le clustering

Le clustering permet de construire une topologie réseau à partir d'un ensemble de nœuds. Pour chaque cluster, il existe un nœud particulier appelé *chef de cluster* choisi grâce à des métriques telles l'énergie restante, la densité du réseau ou l'identifiant du nœud. L'avantage de cette organisation en

clusters est qu'il permet de réaliser certaines tâches (collecte, agrégation, compression, traitement, ou relais) au niveau des chefs de cluster.

Pour assurer une bonne utilisation des ressources du réseau et un prolongement de sa durée de vie, chaque nœud du réseau qui n'est pas chef de cluster doit être au moins dans le voisinage d'un chef de cluster. De plus, tous les nœuds chef de cluster doivent pouvoir former un sous ensemble connecté appelé CDS (Connected Dominating Set) pour le transfert et le relayage des données collectées de leur cluster respectif vers le puits ou centre de collecte (voir figure 1.9).

Dans l'exemple de la figure 1.9, l'ensemble CDS est composé de chefs de cluster (A, D, F) et d'autres types de nœuds appelés *passerelles* qui permettent dans certaines situations de relayer les données collectées par un chef de cluster vers un autre chef de cluster.

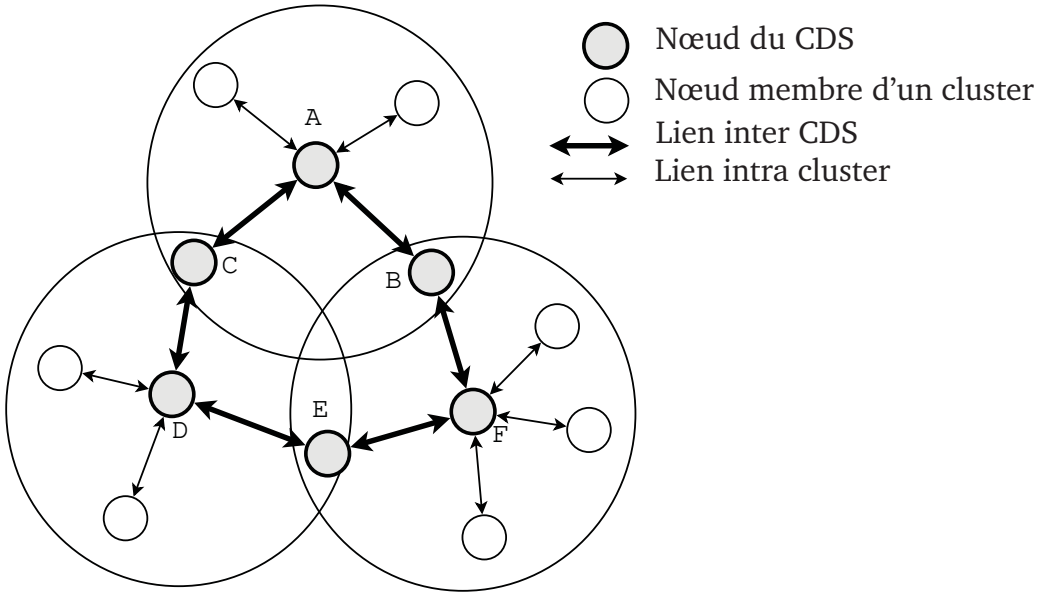



**Fig. 1.9.:** Modèle d'architecture avec l'approche clustering

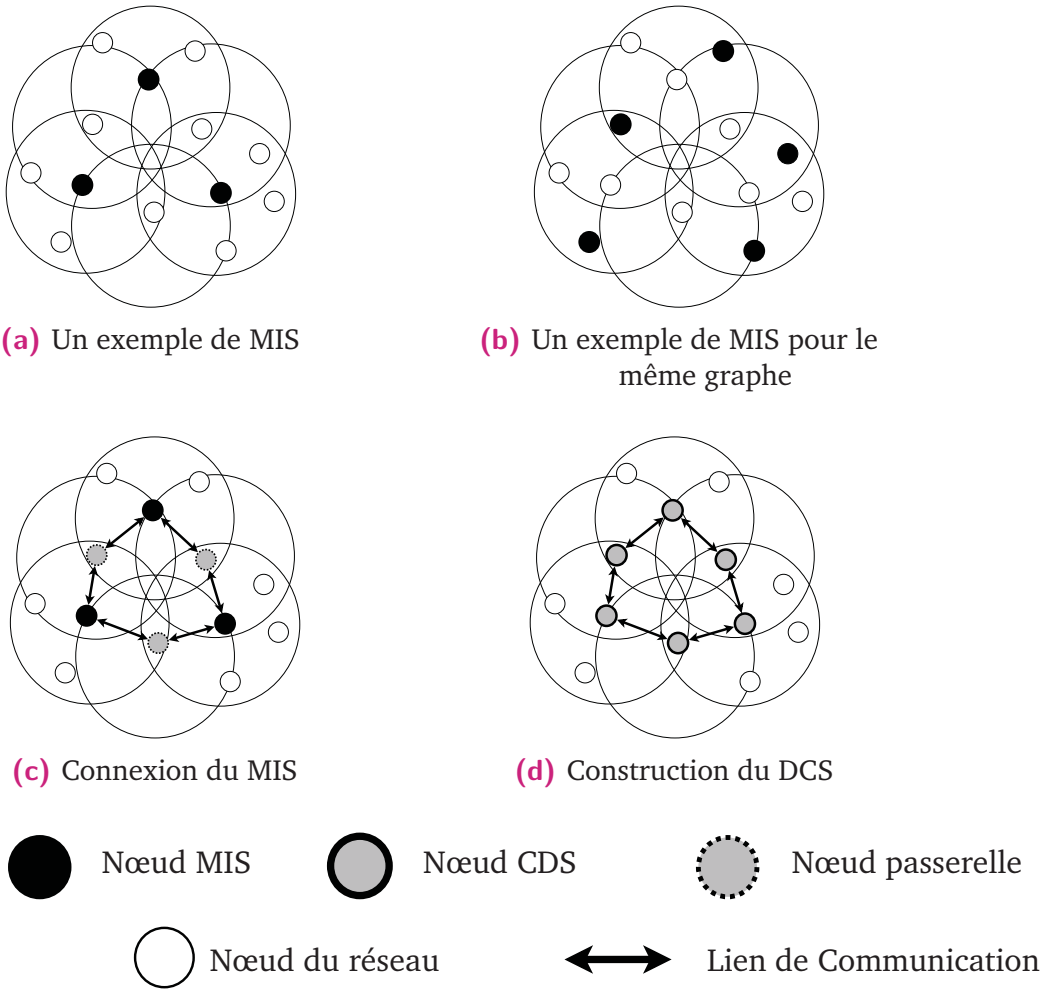
Dans [Jie+02], les auteurs proposent un mécanisme appelé PACDS (Power Aware Connected Dominating Set) basé sur l'approche des CDS. PADCS fonctionne en deux phases pour former la topologie de clusters. La première phase correspond à la formation de l'ensemble des nœuds du CDS. Pour ce faire, chaque nœud effectue une découverte de voisinage et chaque nœud ayant au moins deux voisins non connectés est choisi comme nœud CDS. La deuxième phase consiste à éliminer les nœuds redondants afin de former un

ensemble CDS le plus minimal possible. Le choix de l'ensemble CDS final est basé sur l'identifiant (nodeID) et sur l'énergie résiduelle de chaque nœud. Pour ce faire, les nœuds de l'ensemble CDS, avec la plus petite énergie résiduelle et ayant dans leur voisinage le nœud avec la plus grande nodeID couvrant tous leurs autres nœuds voisins sont éliminés. Afin de résoudre les problèmes de surcharge et ainsi prolonger la durée de vie du réseau, un mécanisme de répartition de charge est ajouté dans PACDS. Ce mécanisme permet de permuter le rôle des nœuds appartenant à l'ensemble CDS dans l'organisation des clusters en fonction de l'énergie résiduelle.

Les auteurs dans [Yua+06b] ont proposé quant à eux un mécanisme appelé ECDS (Energy Efficient Distributed Connecting Dominating Sets). EDCS permet de former un ensemble minimal et optimal de nœuds CDS appelé MCDS (Minimal Connected Dominating Set) comme *backbone* pour le RdCSF. EDCS fonctionne en deux phases. La première phase permet de construire un ensemble maximum indépendant MIS (Maximum Independent Set) qui représente les nœuds du réseau n'ayant pas de lien commun. Le choix de l'ensemble MIS est basé sur l'énergie résiduelle et le nombre de voisins de chaque nœud. Ensuite, dans la deuxième phase, le protocole EDCS détermine les nœuds non MIS, appelés *nœud passerelle*, qui peuvent être utilisés afin de former un ensemble connecté. Ces nœuds passerelles sont choisis en fonction de leur énergie résiduelle et de leur nombre de nœuds MIS voisins. Cet ensemble connecté est appelé ensemble CDS. Cette approche, basée sur le choix des MIS, permet d'avoir un ensemble CDS avec les nœuds ayant la plus grande énergie résiduelle. Afin d'étendre la durée de vie du réseau, EDCS met en œuvre un mécanisme de répartition de charge pour une distribution équitable de la consommation d'énergie des nœuds du réseau. Les schémas de la figure 1.10 représentent les différentes procédures de construction d'un ensemble CDS avec le protocole ECDS dans le but de former des nœuds dominants (détails donnés dans [Yua+06b]).

### 1.3.3 Mécanismes hybrides

D'autres mécanismes combinent les approches d'ajustement de la puissance, d'état du nœud, d'énergie résiduelle et de clustering pour le contrôle de la topologie et la formation des clusters. Parmi ces protocoles, nous pouvons citer le protocole LEACH [Hei+00] et ses dérivés tels que MultiHop-




**(a)** Un exemple de MIS

**(b)** Un exemple de MIS pour le même graphe

**(c)** Connexion du MIS

**(d)** Construction du DCS

**Fig. 1.10.:** Construction du EDCS

LEACH [AB+16], MultiRouting-LEACH [Far+10], LEACH-Selective cluster [Jun+10], etc.

LEACH permet d'organiser le RdCSF en clusters coordonnés par un nœud appelé *chef de cluster*. Le chef de cluster se charge d'allouer des périodes de temps à chaque nœud membre du cluster afin d'éviter les interférences, les collisions et donc les retransmissions de paquets. Chaque chef de cluster se charge aussi, suivant l'application, d'agréger, de compresser ou de fusionner les données reçus des nœuds membres du cluster avant de les retransmettre au puits du réseau. La figure 1.11 montre un RdCSF organisé en clusters composé d'un puits et de trois clusters.

L'activité de LEACH pour l'organisation des nœuds en clusters, la collecte et l'acheminement du trafic au puits est répartie sur plusieurs tours. À chaque tour, LEACH élit un nouveau chef de cluster aléatoirement afin de répartir la charge et l'énergie résiduelle des nœuds et permet de prolonger la durée de vie du RdCSF. Chaque tour de LEACH est divisé en deux phases : La première phase appelée *Setup phase* permet d'élire un chef de cluster aléatoirement en fonction de l'équation (1.4) et de deux critères (le pourcentage de chefs de cluster existant dans le réseau, et le nombre de fois qu'un nœud a été choisi comme chef de cluster). Chaque nœud choisit un nombre aléatoire $u$ entre $0$ et $1$. Si le nombre est plus petit que la valeur du *Threshold* (voir l'équation (1.4)) alors il devient un chef de cluster. Dans l'équation (1.4), $P$ représente le pourcentage de chef de cluster désiré dans le RdCSF, $r$ est le numéro du tour courant et $G$ représente l'ensemble des nœuds qui n'était pas chef de cluster au dernier tour.

$$Th(u) = \begin{cases} \frac{P}{1-P(r\times\frac{1}{P})} & \text{si } n \in G \\ 0 & \text{sinon} \end{cases} \tag{1.4}$$

Chaque chef de cluster diffuse un message dans son voisinage et chaque nœud qui veut devenir membre répond à ce message pour rejoindre le cluster. Lorsqu'un nœud non membre d'un cluster reçoit plusieurs messages de chefs de cluster différents, il répond à celui dont le signal reçu est le plus puissant. Ensuite, chaque chef de cluster calcule la période d'activité qui sera allouée à chacun des nœuds membres de son cluster et le diffuse dans son cluster.

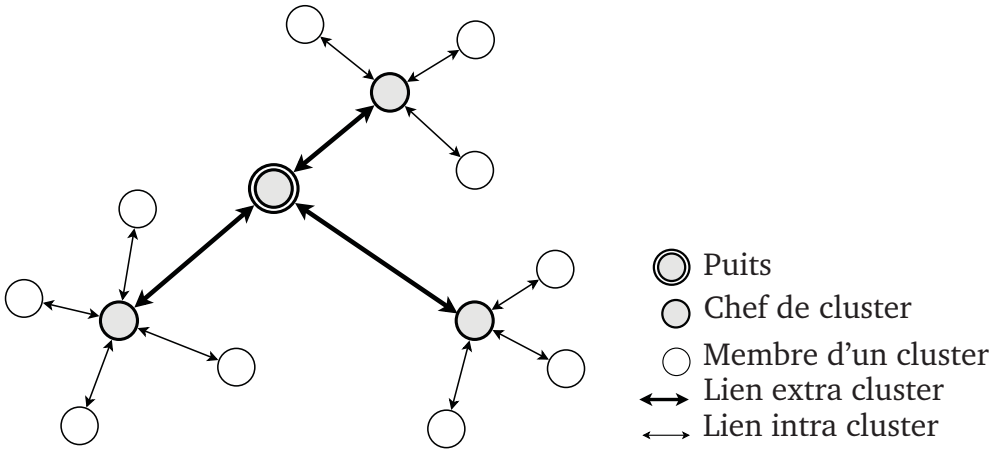



**Fig. 1.11.:** Architecture LEACH

Durant la seconde phase *Steady phase*, chaque nœud envoie ses données à son chef de cluster pendant sa période d'activité et éteint son interface radio en dehors de cette période pour économiser son énergie. Pour la transmission des données collectées au nœud puits, chaque chef de cluster doit être à portée radio du puits. Pour ce faire le chef de cluster est capable d'augmenter la puissance de transmission ce qui a comme conséquence un épuisement rapide de l'énergie des nœuds. À la fin d'un tour, tous les nœuds repassent à la phase de *Setup* pour élire un nouveau chef de cluster parmi les nœuds qui n'ont pas été des chefs de cluster au tour précédent et qui ont le plus petit nombre de sélections en tant que chef de cluster.

Les améliorations de LEACH différent d'une part par la façon d'élire le chef de cluster et d'autre part par le mécanisme d'acheminement des données collectées au puits.

Dans le protocole LEACH-Centralized (LEACH-C) [Hei+02], les auteurs proposent une version centralisée de LEACH. Dans LEACH-C protocole, l'élection des chefs de cluster est contrôlée par le puits du réseau appelé aussi *station de base*. Durant la phase de formation des clusters (*setup phase*), chaque nœud du réseau envoie ses informations de localisation (grâce à un capteur GPS) et son niveau d'énergie résiduelle au puits. Grâce à ces informations reçues, le puits élit un nombre prédéterminé de chefs de cluster parmi ceux ayant la plus grande énergie résiduelle. Cet ensemble de chefs de cluster permet aux autres nœuds du réseau de transmettre leurs données à un chef de cluster tout en minimisant leur énergie consommée. Le puits diffuse, à

tous les nœuds, le message contenant l'identifiant (ID) du chef de cluster de chaque nœud. Lorsqu'un nœud du réseau reçoit ce message, il devient un chef de cluster si l'ID de son chef de cluster est identique à son propre ID de nœud. Dans le cas contraire, le nœud contacte son chef de cluster afin de déterminer sa période d'activité dans son cluster avant de passer en mode sommeil jusqu'à l'heure à laquelle il doit transmettre ses données. La phase de maintenance et de transmission *Steady phase* de LEACH-C est identique à celle LEACH. Cette gestion centralisée et la connaissance globale du réseau par le puits de la formation des clusters permet de garantir une meilleur répartition des chefs de cluster dans le réseau et un même nombre de clusters formés à chaque tour offrant ainsi une réduction d'énergie consommée durant la transmission des données dans les différents clusters.

Dans le protocole MultiRouting-LEACH [Far+10], les auteurs implémentent une organisation (une hiérarchie) en niveau des clusters. Dans cette architecture, les données collectées par les chefs de cluster de deuxième niveau seront acheminées jusqu'au puits en passant par leur chef de cluster respectif de premier niveau. Cette méthode de routage permet d'éviter les liens longues portées entre les chefs de cluster éloignés du puits et ainsi d'économiser au maximum la consommation d'énergie. La phase de construction des clusters débute par une étape de découverte de voisinage où chaque nœud diffuse son énergie résiduelle et son état (chef de cluster, membre de cluster, inconnu). Le nœud qui a la plus grande énergie résiduelle dans son voisinage s'auto élit comme chef de cluster avant de diffuser l'annonce à ses nœuds voisins. Après la phase de construction des clusters, les chefs de cluster s'organisent par niveaux. Le puits diffuse son ID dans son voisinage, chaque chef de cluster qui reçoit le message répond au puits en incluant son propre ID avec une faible puissance de transmission. Le puits établit ainsi la liste des chefs de cluster de premier niveau (ceux qui sont proches), ensuite le puits diffuse un nouveau message incluant la liste des ID des nœuds de premier niveau. Les chefs de cluster n'ayant pas leur ID dans la liste répondent en utilisant un mode de transmission de faible portée. Les nœuds chefs de cluster de premier niveau vont recevoir les messages avant de les relayer au puits. Le puits dresse ainsi la liste des chefs de cluster, leur niveau et désigne le prochain chef de cluster qui servira de relais pour joindre le chef de cluster de niveau inférieur (de $N$ à $N-1$). La figure 1.12 montre une représentation d'une architecture de MultiRouting-LEACH avec deux niveaux d'organisation des clusters.

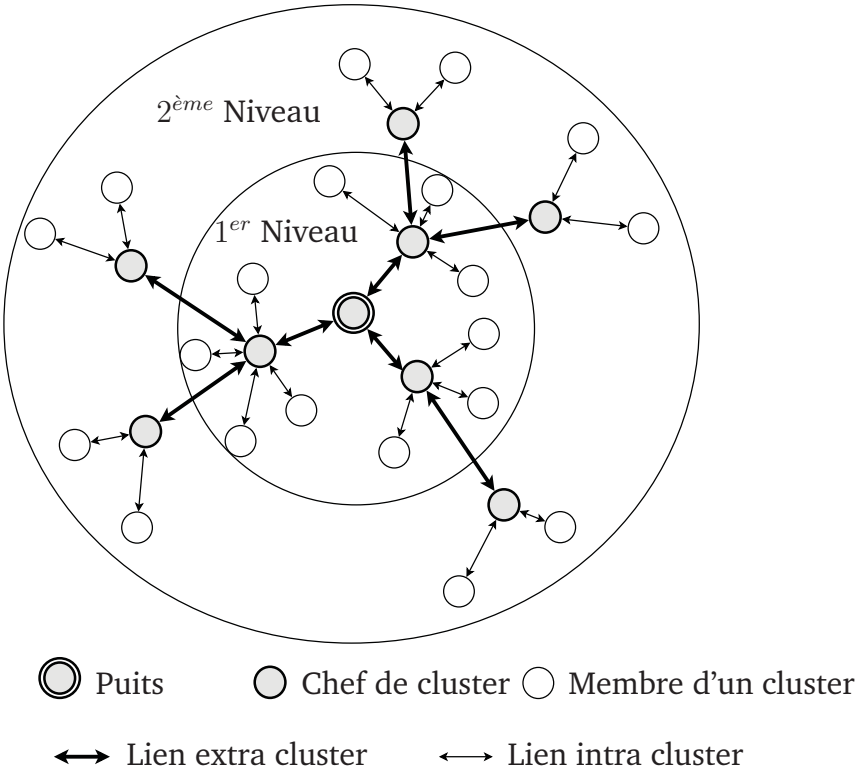



**Fig. 1.12.:** Architecture de MultiRouting-LEACH à deux niveaux

Dans le protocole MultiHop-LEACH [AB+16], les auteurs définissent un seuil d'énergie résiduelle en dessous duquel un nœud ne peut pas devenir un chef de cluster. De plus, deux types de communications multi-saut ont été implémentés : la communication intra-cluster et inter-cluster. Avec la communication intra-cluster, un nœud membre d'un cluster qui n'est pas à portée du chef de cluster peut utiliser des nœuds relais intermédiaires pour le routage des données jusqu'au chef de cluster en fonction de l'énergie résiduelle du ou des nœud(s) relais et de l'énergie consommée sur le chemin jusqu'au chef de cluster (voir la figure 1.13). Pour la communication inter-cluster, un chef de cluster qui veut envoyer ces données au puits peut utiliser d'autres chefs de cluster comme relais. Le choix des chefs de cluster relais, pour la communication inter-cluster se fait en fonction de l'énergie résiduelle et du rapport signal/bruit du nœud. Cette approche du MultiHop-LEACH permet d'éviter les communications longue portée qui consomment beaucoup d'énergie et des retransmissions de paquets. La figure 1.13 montre une architecture de Multihop-LEACH où nous pouvons voir un exemple de communication intra-cluster en multi-saut entre les nœuds membres et leur

chef de cluster. Cette figure montre aussi l'acheminement des données collectées par un chef de cluster vers le nœud puits du réseau.

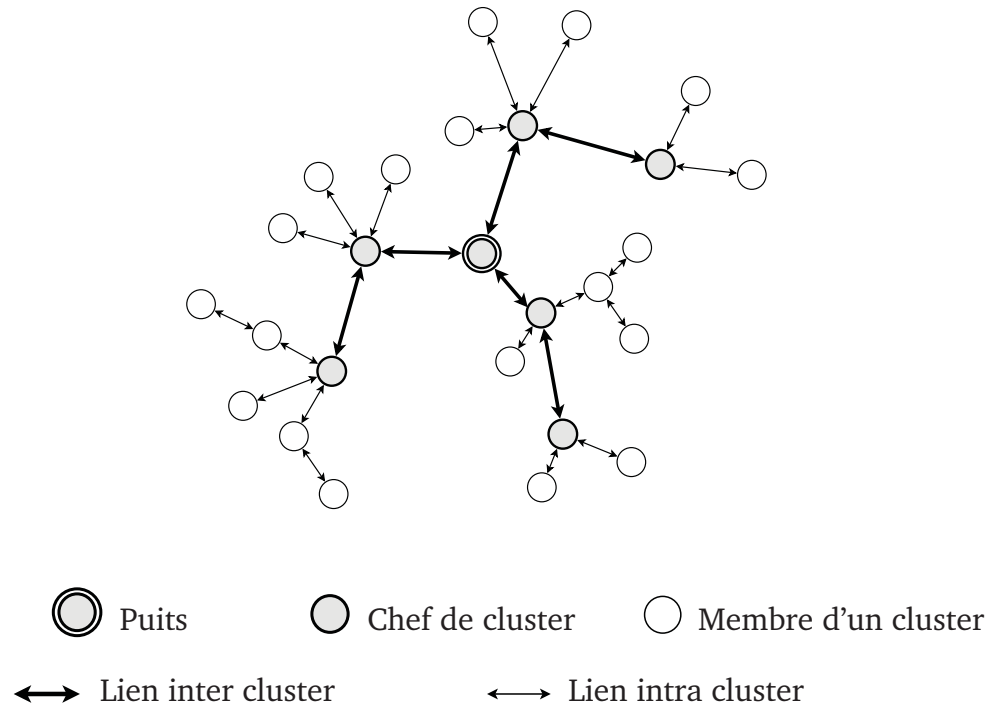



**Fig. 1.13.:** Architecture de MultiHopLEACH

## 1.4 Les réseaux de capteurs sans fil linéaires

Récemment, de nouvelles applications pour les RdCSF ont vu le jour telles que les infrastructures ferroviaires, les ponts, les cours d'eau, les canalisations de gaz et pétrole, les frontières. Ces environnements sont caractérisés par une ou plusieurs portions de lignes avec des embranchements et parcourant plusieurs kilomètres. De ce fait, la recherche dans ce domaine a vu l'émergence d'un nouveau type de topologies de réseaux de capteurs sans fil appelé *réseau de capteurs sans fil linéaire* (RdCSFL ou RdCSF Linéaire) [Jaw+11 ; Pan+08 ; Zim+08]. Un RdCSF Linéaire est un réseau où tous les nœuds de capteurs sont déployés approximativement le long d'une ligne ou de plusieurs lignes reliées. Les RdCSFL ont généralement les caractéristiques suivantes :

— Chaque nœud déployé joue à la fois le rôle de capteur de donnée et de nœud relais,

— La plupart des nœuds ont au moins un voisin dans chaque direction (vers le puits et vers la direction opposée) afin de transmettre les données capturées et de relayer les données venant d'autres nœuds du réseau. Le nombre de voisins de chaque nœud dépend de sa position dans le réseau, de la portée de son signal radio par rapport à la densité des nœuds.

— Le routage des données collectées se fait de proche en proche jusqu'au nœud puits,

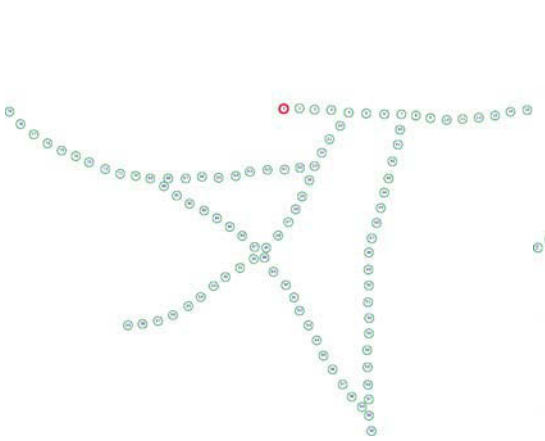
Lorsque le RdCSFL comporte plusieurs lignes reliées, les zones situées au niveau des points de jonction, que nous appellerons dans la suite *zone de jonction* ou *zone de branchement*, ont une grande concentration de nœuds. Par rapport à la portée de leur signal radio et à la distance entre les nœuds, les nœuds situés dans ces zones de branchement ont la particularité d'avoir un grand nombre de voisins.

La figure 1.14 montre des exemples de réseaux linéaires composés de 100 et 200 nœuds composés de plusieurs lignes reliées.

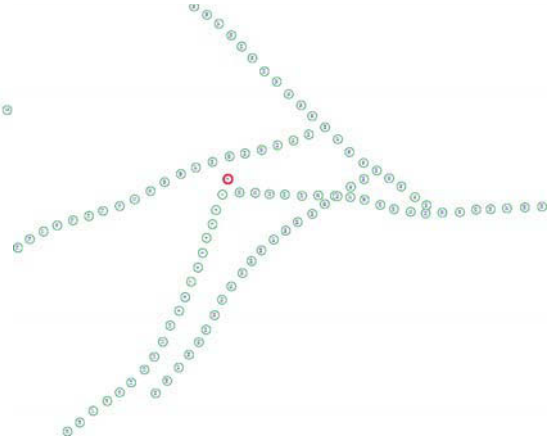
Ces caractéristiques des RdCSFL font que l'acheminement des données subit une latence surtout quand les nœuds sont déployés sur de longues distances. Chaque nœud du réseau transmet ses propres données capturées ainsi que les données reçues de son prédécesseur à son voisin choisi comme prochain saut à destination du puits. Ce mode de fonctionnement occasionne une augmentation du trafic qui empire au fur et à mesure que celui-ci converge vers le puits et donc des débordements de fil d'attente, de la congestion ainsi que des pertes de paquets. Les nœuds proches du puits épuiseront leur énergie plus rapidement que les autres nœuds du réseau car ils sont le plus souvent actifs pour relayer vers le puits la quantité de données importante reçue. Par rapport à la nature du déploiement des nœuds dans le RdCSFL, la perte d'un ou plusieurs nœuds peut rendre une partie du réseau indisponible.

### 1.4.1 Topologies dans les réseaux de capteurs sans-fil linéaires

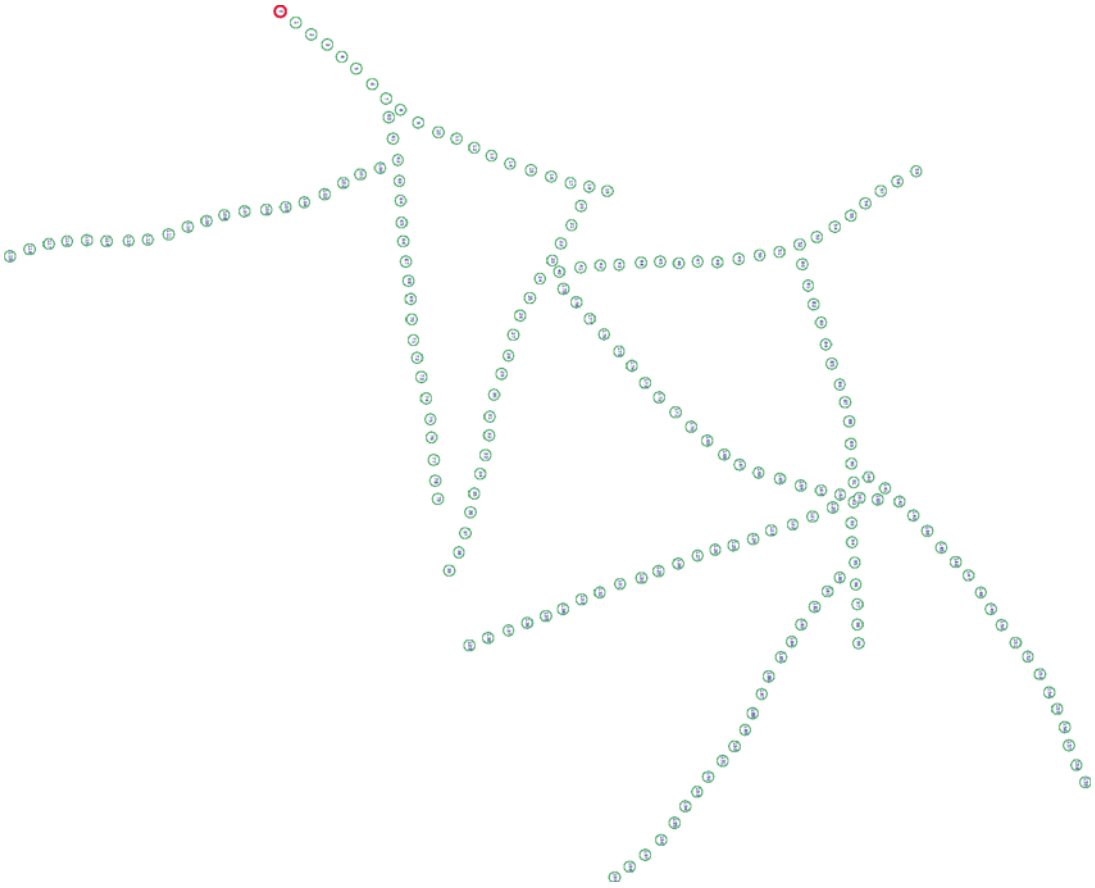
Afin d'assurer une bonne organisation des RdCSFL, plusieurs types de topologies ont été proposées. Cette organisation est dictée par la nature linéaire



**(a)** Topologie de 100 nœuds avec 6 branchements



**(b)** Topologie de 100 nœuds avec 4 branchements



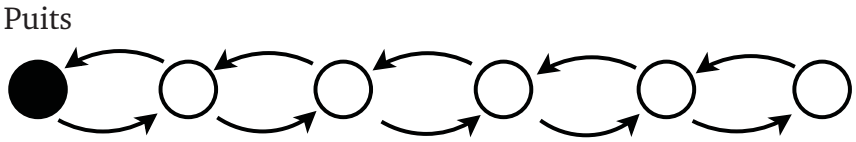
**(c)** Topologie de 200 nœuds avec 9 branchements

**Fig. 1.14.:** Exemples de Topologies Linéaires

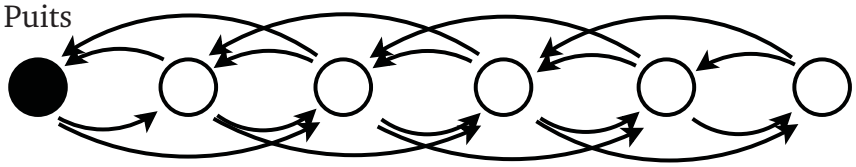
de l'application qui exige le plus souvent un placement des nœuds le long de l'infrastructure linéaire à surveiller.

### Topologies $k$-redondants

Dans les RdCSF linéaires, la connexité des nœuds est primordiale pour le relayage des données et la disponibilité du réseau. Dans [Ndo+14a], les auteurs proposent des architectures $k$-redondantes où chaque nœud de capteurs a $k$ voisins en direction du puits et autant en direction de l'intérieur du réseau. En d'autres termes, un réseau est $k$-redondant si le graphe de voisinage est *$k$-arrête-connexe*. Pour un RdCSF linéaire 1-redondant (voir figure 1.15a), la perte d'un nœud de capteurs peut causer un partitionnement du réseau. Pour $k \geq 2$ (voir figure 1.15b), le RdCSFL fournit une disponibilité et une fiabilité accrues par rapport à la perte de nœuds.




**(a)** Topologie 1-redondante




**(b)** Topologie 2-redondante

**Fig. 1.15.:** Type de topologies RdCSF $k$-redondants

### Topologies hiérarchiques à $N$ niveaux

Dans [Jaw+11], les auteurs proposent une topologie hiérarchique sur une dimension à $N$ niveaux. Dans cette topologie, trois catégories de nœuds de capteurs sont définis en fonction de leurs caractéristiques. Les auteurs utilisent des noms particuliers pour désigner ces nœuds et évoquer le cheminement des données. Les nœuds de Capteurs Simples (NCS) ou BSN (Basic Sensor Node) permettent uniquement de capturer les données des en-

vironnements et de les transmettre aux nœuds relais. Les Nœuds Relais de Données NRD ou DRN (Data Relay Node) assurent la collecte des données transmises par les NCS situés dans leur voisinage à un saut. Cette contrainte suppose que tous les nœuds NCS soient à portée de transmission d'au moins un nœud NRD. Afin d'envoyer les données collecter jusqu'au puits ou Centre de Contrôle et de Traitement des Données (CCTD), les NRD communiquent avec des Nœuds d'Acheminement de Données (NAD) ou DDN (Data Dissemination Node).

La technologie 802.15.4/ZigBee permet l'échange de données entre les nœuds NCS et NRD et entre les nœuds NRD et NAD. De ce fait, la distance entre un NRD et un NCS et entre un NRD et un NAD est conditionnée par les propriétés radio des nœuds (puissance d'émission, sensibilité). Les NAD sont aussi capables de transmettre les données au puits en utilisant une technologie longue portée (cellulaire, satellitaire).

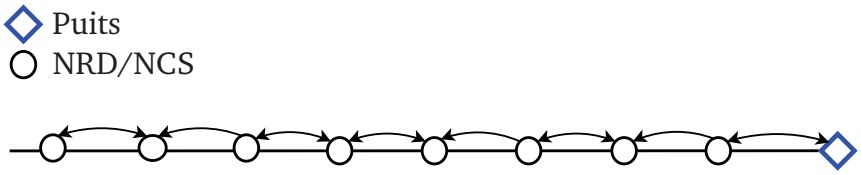



**Fig. 1.16.:** Architecture d'un réseau linéaire à 1-niveau

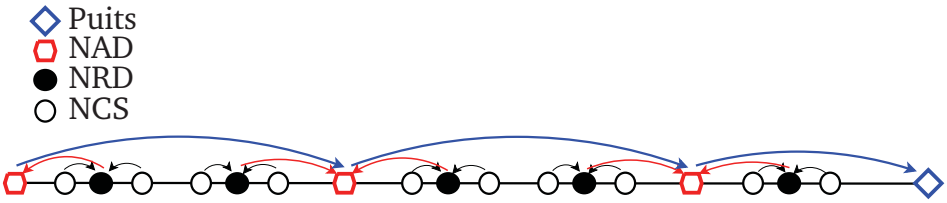



**Fig. 1.17.:** Architecture d'un réseau linéaire à 3-niveaux

Les auteurs de [Jaw+11] décrivent aussi trois types de déploiement possible : (1-niveau, 2-niveaux ou 3-niveaux). Le déploiement linéaire sur 1-niveau est décrit par la figure 1.16 et suppose que la topologie est composée uniquement de NRD jouant à la fois le rôle de capteur et de relais. Dans ce type de déploiement, les données sont relayées saut par saut jusqu'au puits situé le plus souvent à l'une des extrémités du réseau linéaire. Ce qui suppose que chaque nœud doit être à portée de son voisin en direction du puits de collecte.

L'inconvénient de ce type de déploiement est que le réseau linéaire est vulnérable à la perte d'un nœud (épuisement de son énergie, perte de son lien radio) occasionnant ainsi un partitionnement du réseau. La topologie 1-niveau peut occasionner des délais d'acheminement des données dans le cas d'un déploiement sur de longues distances linéaires mais peut engendrer des congestions sur les nœuds au voisinage du puits occasionnant ainsi des pertes de paquets et des débordements de files d'attente. Une solution est de proposer un déploiement linéaire à 2-niveaux où nous avons deux types de nœuds (NCS et NRD).

Avec la topologie 2-niveaux, chaque NRD est responsable de la collecte des données capturées par les NCS de son voisinage avant de les transmettre au puits ou CCTD via un protocole de routage multi-saut à travers les autres NRD déployés. Il existe aussi un déploiement linéaire à 3-niveaux (voir figure 1.17) où nous rencontrons les nœuds NCS, NRD et NAD. Dans ce cas, les nœuds NRD acheminent les données vers le NAD le plus proche par routage multi-saut. Ce dernier se charge à son tour de transmettre les données au puits via une technologie IP, GSM ou Satellitaire.

Le choix d'un type de topologie à déployer dépend fortement de la nature du réseau physique linéaire et de l'application à mettre en œuvre. Pour le déploiement sur une longue distance, la topologie à 3-niveaux permet de réduire considérablement la latence de bout-en-bout, la congestion au voisinage du puits mais aussi la perte de paquets et le débordement de files d'attente grâce aux NRD et NAD qui permettent l'acheminement des données sur de grandes distances en utilisant des technologies longue portée.

### Topologies en arbre de clusters

En ce basant sur la topologie en arbre de clusters de ZigBee, les auteurs de [Pan+08] proposent une topologie en arbre de clusters pour les RdCSF linéaires dans laquelle chaque nœud joue le rôle de capteur et de relais afin d'assurer le routage multi-saut jusqu'au puits situé à l'une des extrémités du réseau. Dans cette topologie en arbre de clusters linéaire, les nœuds sont organisés en clusters logiques représentant chacun un segment linéaire. Chaque cluster est délimité par des nœuds particuliers appelés *Chef de Cluster* (CC) et *Pont de Cluster* (PC) et entre lesquels des nœuds capteurs sont

déployés (voir figure 1.18). Ces nœuds sont *manuellement* choisis au moment du déploiement par son administrateur. Le CC est le nœud du cluster nécessitant le plus petit nombre de sauts pour atteindre le puits et le PC est le nœud nécessitant le plus grand nombre de sauts. L'acheminement des données collectées dans un cluster se fait saut par saut à travers les nœuds d'un même cluster. pour le passage d'un cluster à un autre, seul le PC est autorisé à relayer les données vers le CC du Cluster voisin.

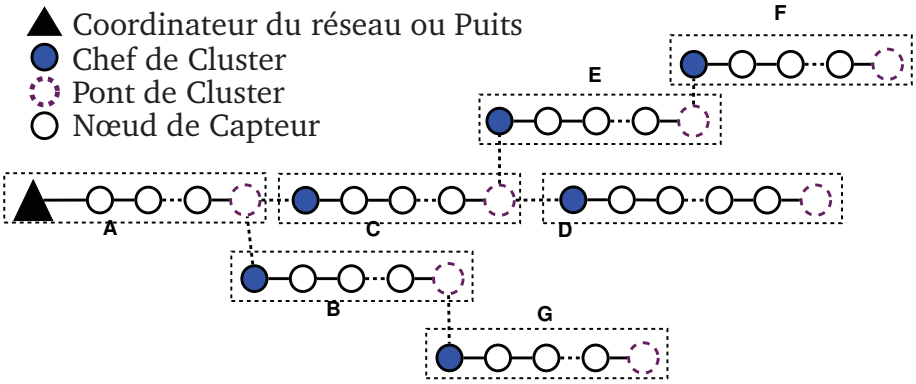



**Fig. 1.18.:** Organisation d'un réseau de clusters de RdCSF Linéaire

## 1.4.2 Protocoles MAC dans les réseaux de capteurs sans-fil linéaires

De nombreux protocoles MAC pour les RdCSF linéaires ont été proposés pour tenter de résoudre les problèmes d'accès au canal, de congestion, d'épuisement des nœuds, de débordement de files d'attente. Nous présentons les protocoles *Dual-Mode Real Time MAC* [Wat+06], *Directional Scheduled MAC (DiS-MAC)* [Kar+09], LTB-MAC (Linear Token-Based MAC) [Ndo+16] qui nous semblent les plus représentatifs.

### Dual-Mode Real Time MAC

*Dual-Mode Real Time MAC* [Wat+06] assure la livraison des messages avant un certain délai en fournissant deux modes de communication : Un mode *protégé* pour des communications fiables sans collision et un mode *non-protégé* au cas où des collisions sont peu probables dans des RdCSF linéaires

où les nœuds sont aléatoirement déployés. Ce protocole est basé sur la position des nœuds par rapport au puits obtenue grâce aux cordonnées GPS relevées pendant le déploiement.

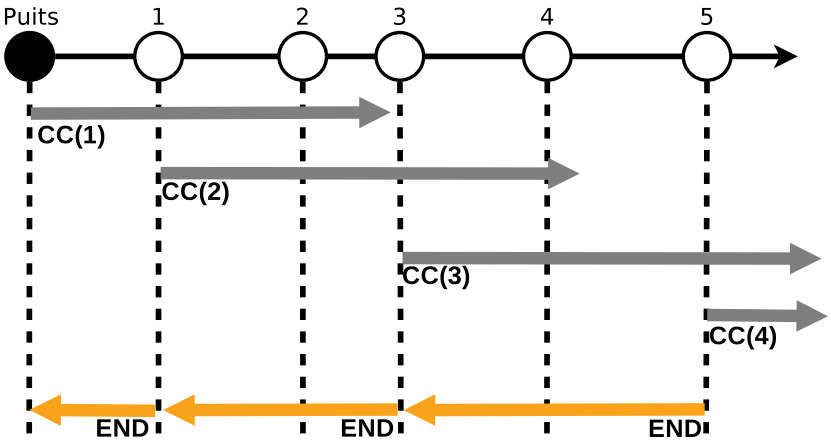
Durant la phase d'initialisation, le protocole regroupe les nœuds du réseau en *cellules*. Le nœud puits commence par émettre un message d'initialisation de cellules appelé *CC(*1*)*, créant ainsi la première cellule (voir figure 1.19a). Chaque nœud qui reçoit un *CC(i)*, avec $i$ le numéro du cellule, initialise un temporisateur appelé $backoff_{init}$ pendant lequel il stocke et comptabilise tous les messages *CC(i)* reçus. À la fin du temporisateur, si le nœud n'a reçu qu'un seul message *CC(i)*, il crée une nouvelle cellule $i+1$ et émet un message *CC(*$i+1$*)*. Si le nœud a reçu plusieurs messages *CC*, exemple *CC(i)* et *CC(*$i+1$*)*, il crée la cellule $i+2$ et émet un message *CC(*$i+2$*)*. Tout en envoyant le message *CC*, le nœud déclenche un temporisateur $timer_{last}$. Si à l'expiration de ce temporisateur, il n'a pas reçu d'autres messages *CC*, le nœud considère qu'il est le dernier du réseau linéaire et émet un message *END* qui est relayé par les nœuds intermédiaires en direction du puits. À ce stade, chaque nœud connaît sa cellule d'appartenance $I$, sa position $R$ dans la cellule ainsi que sa position absolue $A$.

$$backoff_{init} = \frac{A - A_{émetteur}}{V_{init}} \tag{1.5}$$

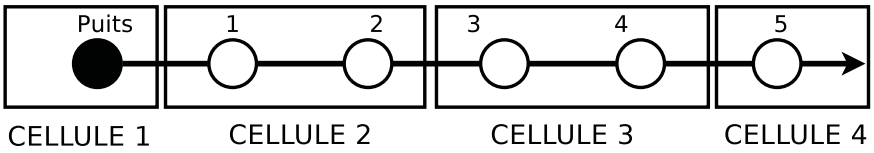
$$V_{init} \leq \frac{dist_{min}}{\frac{longueur_{signalisation}}{BP} + T_{RxTx}} \tag{1.6}$$

Dans l'équation (1.5), A et $A_{émetteur}$ représentent respectivement la position absolue du nœud récepteur et émetteur. Dans l'équation (1.6),$V_{init}$ représente la vitesse en $m/s$ des temporisateurs expirant dans le réseau et est calculée en fonction du temps de transmission d'un message et du temps de passage de l'état de transmission à l'état de réception. $longueur_{signalisation}$ représente la taille en bits des messages $CC$ et $END$, $BP$ est la bande passante du support et $dist_{min}$ est la distance minimale entre les nœuds du réseau.

La figure 1.19 montre un exemple de création de cellules où le puits appartient à la cellule $1$, les nœuds $1$ et $2$ à la cellule $1$, les nœuds $3$ et $4$ à la cellule $3$ et enfin le nœud $5$ à la cellule $4$.




**(a)** Phase d'initialisation des cellules selon [Wat+06]




**(b)** Formation des cellules

**Fig. 1.19.:** Mécanisme de création de cellules

À la fin de la phase d'initialisation, le réseau linéaire fonctionne en premier lieu en mode *non protégé* qui offre un débit de données optimal transféré jusqu'au puits. Dans ce mode de fonctionnement *non protégé*, qui n'est pas basé sur l'organisation en cellule du réseau, chaque nœud qui reçoit un message *ALARME* (message de données) d'un autre nœud, démarre un temporisateur inversement proportionnel à sa distance par rapport au nœud émetteur de l'*ALARME*. À la fin du temporisateur, s'il ne reçoit pas un autre message *ALARME*, ce nœud est choisi comme nœud relais pour les messages de données. La figure 1.20 montre un fonctionnement en mode non protégé où les nœuds 1 et 3 sont utilisés comme nœuds relais pour l'acheminement des données venant du nœud 5.

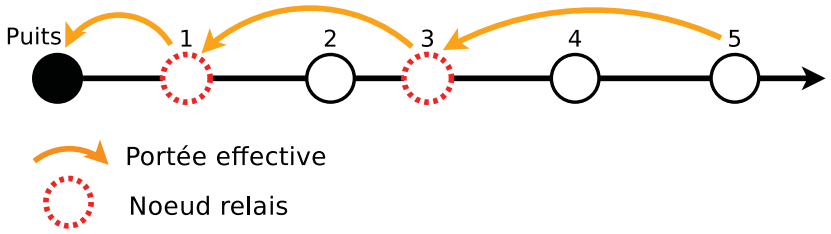



**Fig. 1.20.:** Communication en mode non protégé

Durant la communication en mode *non protégé*, lorsqu'un nœud n'entend pas son message ré-émis après un temps donné ou bien que sa couche physique détecte des interférences, il suppose qu'il y a eu collision. Alors, il passe en mode *protégé*, et émet un message *JAM*. Chaque nœud qui reçoit le message *JAM*, bascule en mode *protégé* avant d'émettre un autre *JAM*. À la fin du processus, tout le réseau bascule en mode *protégé*. Le mode *protégé* garantit une communication sans collision en se basant sur l'organisation en cellules définie durant la phase d'initialisation. Seul le puits peut décider du basculement du mode *protégé* vers le mode *non protégé* en se basant sur le débit de données reçues.

En mode protégé, chaque nœud de la cellule $cell(i)$ voulant émettre, fait une demande de réservation ($RES$) de $5$ cellules maximum en direction du puits ou toutes les cellules qui le séparent du puits si la cellule du nœud émetteur est proche de celle du puits ($cell(1)$). C'est à dire la cellule $cell(i)$ fait une demande de réservation à $cell(i-1)$ qui à son tour fait la demande à $cell(i-2)$. Ainsi un message $RES$ est envoyé de nœud en nœud dans une même cellule et de cellule en cellule jusqu'à la $cell(i-5)$. Le dernier nœud élu de $cell(i-5)$ émet un accusé *ACK* en mode *non protégé*. Lorsque le nœud émetteur du message de réservation $RES(1)$ reçoit le message *ACK*, il démarre la transmission de ses données.

La figure 1.21 présente un exemple de fonctionnement en mode protégé. Le nœud $5$, pour acheminer ses données, fait une demande de réservation de cellules (Cell(4), Cell(3), Cell(2), Cell(1)). L'envoi de l'accusé *ACK* par le puits au nœud $5$ se fait en mode non-protégé, et les nœuds $2$ et $4$ sont ici utilisés comme nœuds relais.

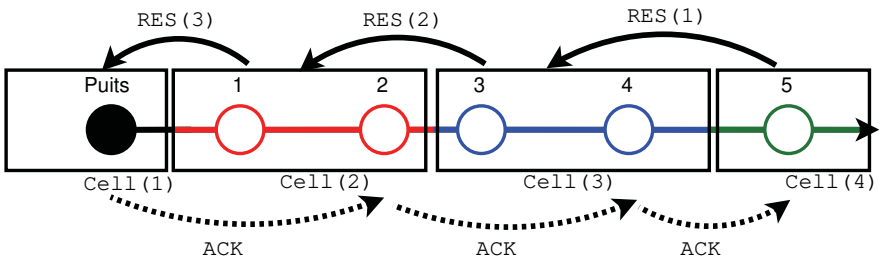



**Fig. 1.21.:** Communication en mode protégé

### Directional Scheduled MAC (DiS-MAC)

*Directional Scheduled MAC (DiS-MAC)* [Kar+09] est un protocole MAC de Rendez-Vous qui garantit une communication sans collisions entre des nœuds synchronisés d'un réseau linéaire pour la surveillance des autoroutes. Dans DiS-MAC, chaque nœud est équipé d'une antenne directionnelle, avec un grand gain dans une direction donnée et un gain faible dans la direction opposée. Dis-MAC divise l'accès au canal en deux phases $P1$ et $P2$ de durée respective $T_1$ et $T_2$. Par défaut, $T_1 = T_2 = T$. Lorsque le réseau linéaire de $n$ nœuds est dans la phase $P1$, tous les nœuds de profondeur $2i - 1$ sont autorisés à transmettre durant un temps $T_1$ (voir figure 1.22). À la phase $P2$, les nœuds à la position $2i$ vont transmettre à leur tour pendant un temps $T_2$. Ce qui fait que durant un cycle $C = T_1 + T_2 = 2T$, tous les nœuds passeront par deux états : réception et transmission.

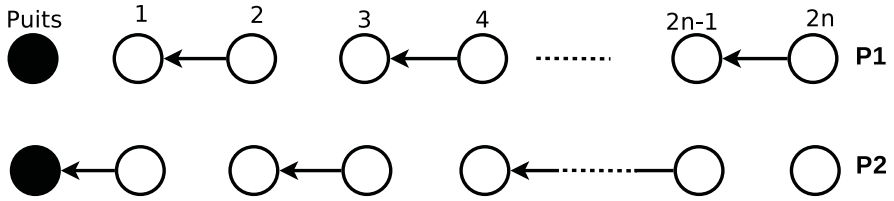



**Fig. 1.22.:** Fonctionnement de DiS-MAC

### Long-Chain MAC (LC-MAC)

*Long-Chain MAC (LC-MAC)* [Fan+11] est un autre protocole linéaire MAC basé sur les techniques de détection de la position des nœuds et sur la réservation à l'avance des nœuds relais. Dans LC-MAC, tous les nœuds sont placés à égale distance et sont à portée de transmission l'un de l'autre. À l'initialisation du réseau de $n$ nœuds, LC-MAC démarre une phase de *détection des positions des nœuds* où chaque nœud relais fait une découverte de voisinage. Tout nœud avec un seul voisin est alors considéré comme un nœud d'extrémité du réseau linéaire et est noté $R_n$ (voir figure 1.23). Ensuite, le nœud $R_n$ se charge d'émettre un message *LDP* (Location Detect Package) incluant les adresses de ses voisins. Chaque nœud voisin relais qui reçoit le message *LDP*, y ajoute à son tour les adresses de ses voisins et le retransmet à un autre voisin relais. Le message *LDP* est relayé nœud par nœud jusqu'au puits

qui ajoute à son tour les adresses de ses nœuds voisins avant de le retransmettre via le chemin retour du message. À la fin de la phase de détection des positions, chaque nœud connaît sa position (nombre de sauts) par rapport au puits et au nœud $R_n$.

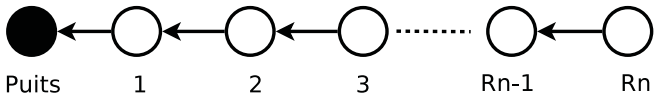



**Fig. 1.23.:** Topologie logique de LC-MAC

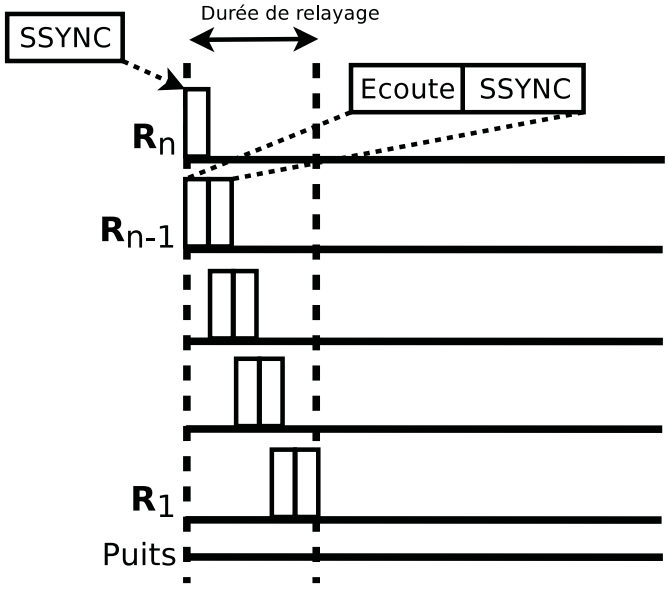



**Fig. 1.24.:** Fonctionnement du relais SSYNC

Après la phase de détection des positions, LC-MAC passe par une phase de *synchronisation* où chaque nœud planifie ses périodes de réveil appelées *SWS* (Staggered Wakeup Schedule) pour le relais du message de synchronisation *SSYNC* (Super SYNC) émis par le nœud $R_n$ à destination du puits. Ces périodes SWS sont définies en fonction de la position du nœud dans le réseau linéaire et de la durée de transmission d'un message SSYNC ($T_S$). La période SWS de chaque nœud de position $i$ compris entre $n-1$ à $1$ est composée d'un temps d'écoute pour recevoir le SSYNC du nœud relais $i+1$ suivi d'un temps de relais où le SSYNC reçu est retransmis au nœud relais à

la position $i-1$ (voir figure 1.24). La durée totale du relayage du mesage SSYNC dans un réseau linéaire de $n$ nœuds est calculé par l'équation (1.7)

$$T_{relayage} = T_S \times n \tag{1.7}$$

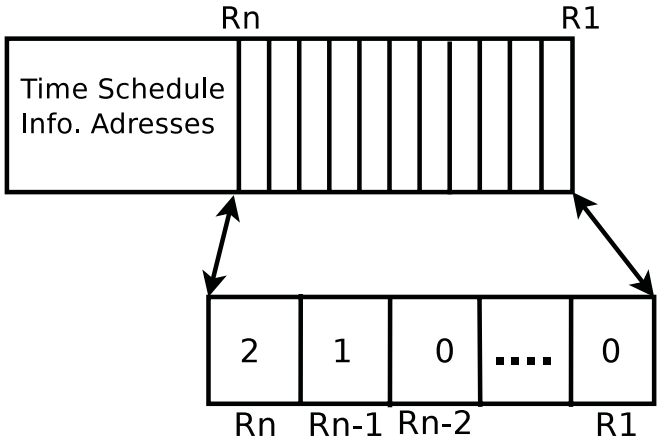



**Fig. 1.25.:** Message SSYNC

Le message SSYNC (voir figure 1.25) permet de définir les périodes de sommeil des nœuds et les nombres de paquets de données à envoyer par chaque nœud relais. Grâce à ces informations contenues dans le message SSYNC, chaque nœud relais calcule la période de réveil et la durée de transmission des données par rapport au nombre de paquets de données à relayer.

### Linear Token-Based MAC Protocol (LTB-MAC)

Dans [Ndo+16], les auteurs proposent un protocole MAC pour les réseaux linéaires basé sur des jetons circulants appelé LTB-MAC (Linear Token-Based MAC). Les auteurs définissent une topologie linéaire où tous les nœuds ont des capacités de capteurs et de relais de données. Dans cette topologie linéaire, le puits est situé à l'une des extrémités du réseau. Le nœud situé à l'autre extrémité du réseau est appelé *Token Allocator* et il se charge de générer le *Jeton* (Token). Le Jeton contient des informations temporelles sur les périodes d'activité et de sommeil des nœuds du réseau. La période d'activité est segmentée en trois périodes : Une période de réception comprenant deux étapes à savoir la réception du trafic descendant $T_0$ et la réception du jeton $T_0'$. Une période de transmission composée de trois étapes : l'émis-

sion des données descendantes (trafic destiné aux nœuds proches du Token Allocator) durant un temps $T_1$, puis l'émission pendant un temps $T_2$ des données montantes (trafic destiné au puits) et enfin la transmission du jeton au prochain nœud relais. Une période de réception des données descendantes durant un temps $T_3$. Après la période d'activité, chaque nœud passe en mode sommeil durant un temps $T4$. La figure 1.26 présente les différentes périodes d'un nœud dans LTB-MAC.

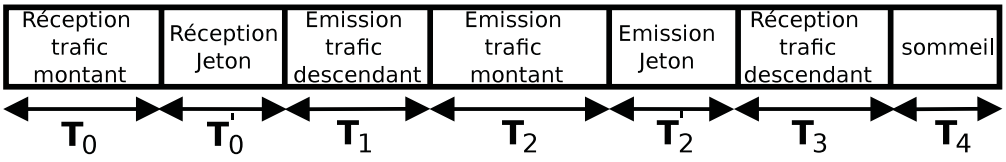



**Fig. 1.26.:** États d'un nœud dans LTB-MAC

## 1.5 Techniques d'adressage dans les RdCSF Linéaires

Dans les réseaux de capteurs sans-fil, les nœuds de capteurs utilisent des adresses logiques pour communiquer entre eux et relayer les données jusqu'au puits. Ces adresses peuvent être attribuées de façon hiérarchique ou stochastique avec dans ce dernier cas des mécanismes de détection et de gestion des conflits afin d'assurer l'unicité des adresses [All08]. Dans le type d'adressage stochastique, l'utilisation d'un protocole de routage tel que *AODV* [Per+03] est nécessaire pour assurer l'acheminement du trafic jusqu'au puits. Les adresses des nœuds peuvent aussi être attribuées de manière hiérarchique. Cette technique est adaptée aux topologies en arbre et permet de faciliter l'acheminement du trafic à destination du puits ou d'un autre nœud du réseau avec l'utilisation d'un protocole de routage très simple basé sur des calculs sur les adresses. Dans la suite, nous présentons les différentes techniques d'adressage pour les RdCSF. Nous commençons par le mécanisme d'adressage du standard ZigBee pour les réseaux de capteurs sans-fil en arbre et nous présentons par la suite en détaillant les mécanismes d'adresses spécifiques aux RdCSF linéaires.

### 1.5.1 Mécanisme d'adressage distribué ZigBee

Le standard ZigBee [All08] fournit un mécanisme de formation, d'adressage et de routage distribué [All08 ; Cun07] pour les topologies en arbre ZigBee. Dans la partie 1.2.1, nous avons présenté la topologie en arbre ZigBee qui peut être composée de trois types de nœuds :

— Un seul nœud Coordinateur pour la gestion du réseau,

— Des nœuds routeurs ZigBee qui sont utilisés pour le relayage des données et la capture de données. Ces nœuds peuvent avoir d'autres nœuds routeurs ZigBee et des nœuds simples comme nœuds fils,

— Des nœuds feuille ou des nœuds simples ZigBee utilisés uniquement pour la capture de données. Ces nœuds n'ont pas la possibilité d'avoir d'autres nœuds fils.

ZigBee implémente le mécanisme distribué d'assignation d'adresses appelé *DAAM* (Distributed Addressing Assignment Mechanism). *DAAM* se base sur les paramètres initiaux du réseau en arbre à savoir le nombre maximum de nœuds fils par nœud routeur ZigBee ($C_m$), le nombre maximum de nœuds routeurs ZigBee fils ($R_m$) par nœud routeur ZigBee et la profondeur maximum du réseau en arbre ($L_m$). Pour assurer une adressage hiérarchique et une unicité des adresses des nœuds, *DAAM* utilise une fonction appelée $Cskip$ (voir équation (1.8)) qui permet de calculer la taille de l'espace d'adressage alloué à chaque sous-arbre du cluster en fonction de la profondeur $d$ du nœud.

$$Cskip(d) = \begin{cases} 1 + Cm \times (Lm - d - 1) & \text{, si } Rm = 1 \\ \frac{1+Cm-Rm-Cm\times Rm^{Lm-d-1}}{1-Rm} & \text{, sinon} \end{cases} \tag{1.8}$$

Les adresses réseaux ZigBee étant codées sur $16$ bits, pour un réseau d'arbres de clusters donné, l'espace d'adressage maximum allouable aux nœuds (Amax) représenté dans le tableau 1.1, dans la limite des $2^{16}$ adresses soit $65536$ adresses, est obtenu grâce à l'équation (1.9).

$$Amax = Cskip(0) \times Rm + Cm - Rm \tag{1.9}$$

| Paramètres de l'arbre de cluster | | | Espace d'adresses |
|---|---|---|---|
| Cm | Rm | Lm | Adresses disponibles |
| 1 | 1 | 65536 | 65536 |
| 2 | 2 | 15 | 65534 |
| 3 | 3 | 9 | 29523 |
| 4 | 4 | 7 | 21844 |

**Tab. 1.1.:** Exemple d'Espace d'adressage disponible avec DAAM

Lorsqu'un nœud routeur ZigBee rejoint le réseau d'arbres de clusters et devient le $n^{ieme}$ nœud fils routeur d'un nœud parent à la profondeur $(d)$, son adresse est calculée avec l'équation (1.10).

$$\begin{aligned} A_{n\triangleleft ud\ routeur}(n) &= A_{parent} + 1, && \text{si } n = 1 \\ A_{n\triangleleft ud\ routeur}(n) &= A_{parent} + (n-1) \times Cskip(d), && \text{si } n > 1 \end{aligned} \tag{1.10}$$

S'il s'agit du $n^{ieme}$ nœud simple ZigBee, son adresse est obtenue avec l'équation (1.11)

$$A_{n\triangleleft ud\ simple}(n) = A_{parent} + Rm \times Cskip(d) + n \tag{1.11}$$

Pour un réseau avec comme paramètres $C_m = 5$, $R_m = 3$ et $L_m = 3$, le tableau 1.2 montre les valeurs de $Cskip$ calculées en fonction de la profondeur $d$.

| **Paramètres :** $C_m = 5$, $R_m = 3$ **et** $L_m = 3$ | |
|---|---|
| **Profondeur du réseau** $(d)$ | **Valeur de** $Cskip(d)$ |
| 0 | 66 |
| 1 | 21 |
| 2 | 6 |
| 3 | 1 |
| 4 | 0 |

**Tab. 1.2.:** Exemple d'adressage avec $Cskip$ en fonction de la profondeur

La figure 1.27 montre une répartition des adresses conformément au paramètres du réseau du tableau 1.2.

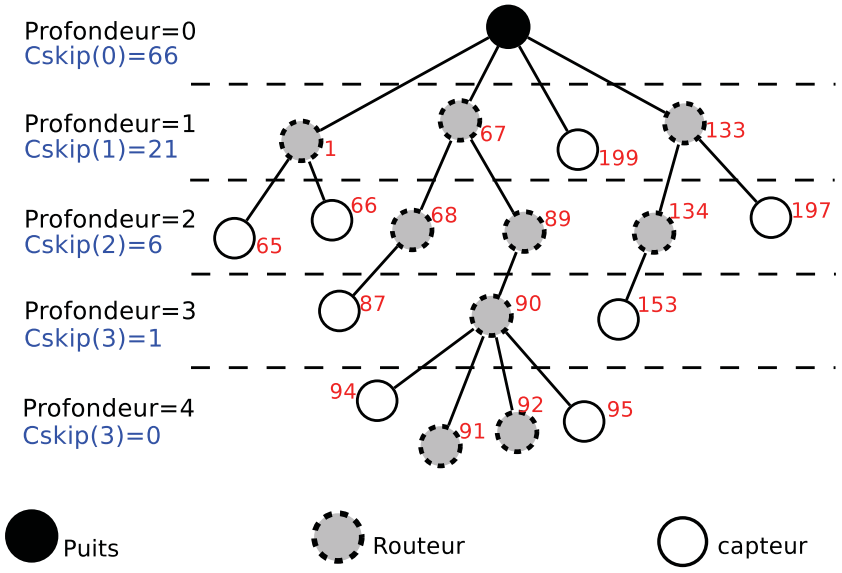



**Fig. 1.27.:** Répartition des adresses dans un réseau en arbre de cluster Zigbee

ZigBee propose avec *DAAM* un mécanisme de routage hiérarchique qui ne nécessite pas d'échanges de tables de routage et qui se base sur la fonction $Cskip$. À la réception d'un paquet de donnée, chaque nœud routeur avec l'adresse $A$ vérifie en premier lieu si l'adresse de destination $D$ est un de ses nœuds fils. Dans ce cas, le paquet de données est directement envoyé au nœud de destination. Sinon, le nœud routeur vérifie si la destination appartient à l'un de ses sous-arbres grâce à la condition (1.12). Dans ce cas, l'adresse du nœud prochain saut $N$ est déterminé grâce à l'équation (1.13). Dans le cas contraire, le paquet est transmis au nœud parent.

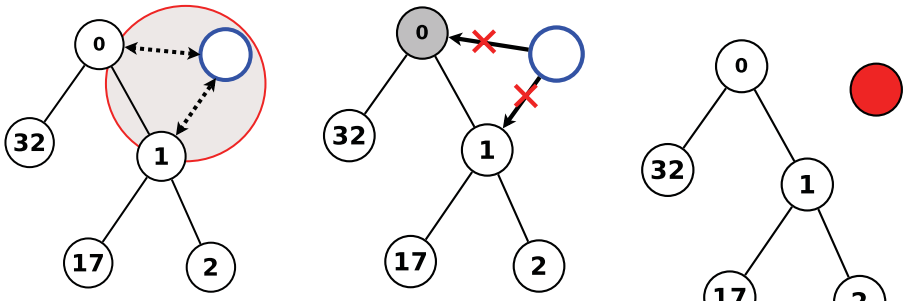
$$A < D < A + Cskip(d) \tag{1.12}$$

$$N = A + 1 + \left[\frac{D - (A+1)}{Cskip(d)}\right] \times Cskip(d) \tag{1.13}$$

Dans le mécanisme d'adressage DAAM, les paramètres $C_m$, $R_m$ et $L_m$ sont définis au début de la planification du réseau c'est à dire avant son déploiement. Ce choix à priori ne permet pas de prédire les bonnes valeurs des paramètres par rapport à la nature de l'environnement de déploiement des nœuds de capteurs et par rapport à la la topologie souhaitée ni d'étendre ultérieurement le réseau. À cela s'ajoute le problème de nœuds orphelins [Pan+09] causé par la manière dont un nouveau nœud s'associe à un nœud de l'arbre. Nous appelons nœud orphelin, un nœud non associé qui n'a pas réussi à ob-

tenir une adresse logique à cause des limitations des paramètres $C_m$, $R_m$ et $L_m$ de l'arbre de cluster.

La figure 1.28 montre un exemple où la tentative d'association du nœud $5$ se solde par un échec. Cet échec est dû aux limitations du réseau ($C_m = 2$,$R_m = 2$,$L_m = 5$) et que ces deux voisins ( $0$ et $2$) ont chacun deux nœuds fils. Par conséquent, le nœud $5$ n'arrive pas à joindre le réseau, ni à obtenir une adresse logique.




**(a)** Requête d'association **(b)** Echec d'association **(c)** Nœud orphelin

**Fig. 1.28.:** Problème de DAAM avec les paramètres $C_m = 2$,$R_m = 2$,$L_m = 5$

Le protocole ZigBee souffre de limitations lorsqu'il est choisi pour les environnements linéaires. En effet, pour des RdCSF linéaires ayant au moins deux branches, ZigBee ne peut adresser les nœuds que jusqu'à la profondeur $Lm = 15$ avec les paramètres $Cm = 2$ et $Rm = 2$. De plus, 15 sauts est très insuffisant pour couvrir un RdCSF linéaires. Pour résoudre ces manquements, des protocoles adaptés aux réseaux linéaires sont proposés pour permettre l'adressage de beaucoup plus de nœuds de capteurs et même au delà des $15$ sauts.

### 1.5.2 Mécanisme Distributed Borrowing Addressing Scheme for Zigbee WSN (DIBA)

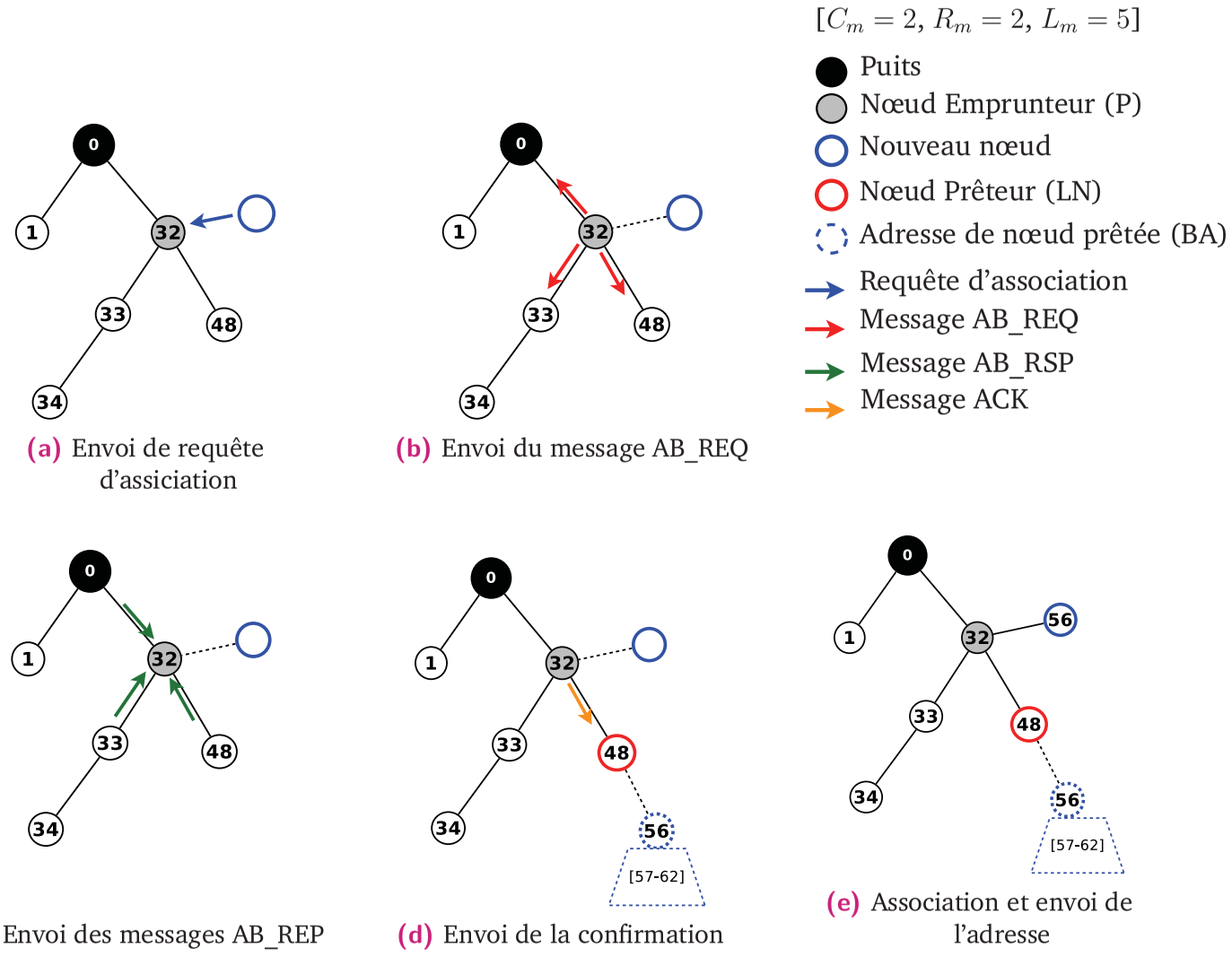
Pour résoudre les problèmes d'apparition des nœuds orphelins, les auteurs de [Par+09] proposent un mécanisme d'adressage basé sur DAAM qui permet d'emprunter des blocs d'adresses non utilisés par un autre nœud du réseau. Chaque nœud routeur du réseau commence par la diffusion d'une balise contenant le nombre d'adresses disponibles appelé *AAC (Available Address Count)*. Lorsqu'un nouveau nœud *N* veut intégrer le réseau, il choisit

un nœud parent (P) à portée de transmission possédant la valeur du *AAC* la plus grande. Le nœud *P* attribue au nouveau nœud une adresse parmi son bloc d'adresses disponibles appelé *AA (Available Address)*. Dans le cas où le nœud parent (P) n'a plus d'adresses disponibles ou si sa profondeur est égale à $L_m$ alors *DIBA* met en œuvre un mécanisme d'emprunt de bloc d'adresses.

Pour ce faire, le nœud *P* diffuse une requête d'emprunt *AB_REQ* (Address Borrowing Request). Lorsqu'un nœud voisin prêteur d'adresses appelé *LN (Lender Node)* et ayant une adresse $Addr_{LN}$ reçoit le message AB_REQ et que le nombre d'adresses prêtés *nbre_BA* est inférieur à $Rm$ alors *LN* calcule l'adresse à prêter *BA (Borrowed Address)* grâce à l'équation (1.14) et répond par un message *AB_RSP* contenant *BA* et *AAC*, et la profondeur $d$.

$$BA = \begin{array}{c} \text{Si } num\_BA < Rm \\ Addr_{LN} + (Rm - num\_BA) \times Cskip(d) + 1, \end{array} \tag{1.14}$$

*P* choisit parmi les réponses *AB_RSP* reçues, l'adresse *BA* avec la plus grande valeur *AAC*. Si *P* reçoit deux *BA* venant de nœuds différents avec la même valeur de *AAC*, alors la *BA* ayant la plus grande valeur est choisie. Puis *P* attribue au nouveau nœud *N* l'adresse *BA* ainsi que le bloc $[BA, BA+Cskip(d)-1]$ pour de futures associations ($d$ étant la profondeur du nœud LN).




**(a)** Envoi de requête d'assiciation

**(b)** Envoi du message AB_REQ

**(c)** Envoi des messages AB_REP

**(d)** Envoi de la confirmation

**(e)** Association et envoi de l'adresse

**Fig. 1.29.:** Différentes étapes de DIBA

La figure 1.29 montre un exemple de fonctionnement de *DIBA* où un nouveau nœud essaie de joindre le réseau. Le nœud $32$, ayant atteint son nombre maximum de fils, déclenche le processus d'emprunt d'adresses. Après échange de messages *AB_REQ* et *AB_RSP*, le nœud $32$ choisit l'adresse proposée par le nœud $48$ car il a comme $AAC = 2$ et la valeur $BA$ proposée est $56$. Le nœud $32$ fournit alors l'adresse $56$ au nouveau nœud ainsi que son bloc $[57 - 62]$.

Les auteurs de *DIBA* ont aussi proposé un mécanisme de routage basé sur *DAAM* afin de continuer à assurer l'acheminement des données. Chaque nœud maintient une table de routage avec les informations de ses nœuds fils.

Table de routage du nœud $32$

| | Adresse | Prêteur | Emprunteur |
|---|---|---|---|
| Parent | 0 | - | - |
| Fils 1 | 33 | - | - |
| Fils 2 | 48 | - | - |
| Fils 3 | 56 | 48 | - |

Table de routage du nœud $48$

| | Adresse | Prêteur | Emprunteur |
|---|---|---|---|
| Parent | 32 | - | - |
| Fils 1 | 56 | - | 32 |

**Tab. 1.3.:** Tables de routage de quelques nœuds de la figure 1.29

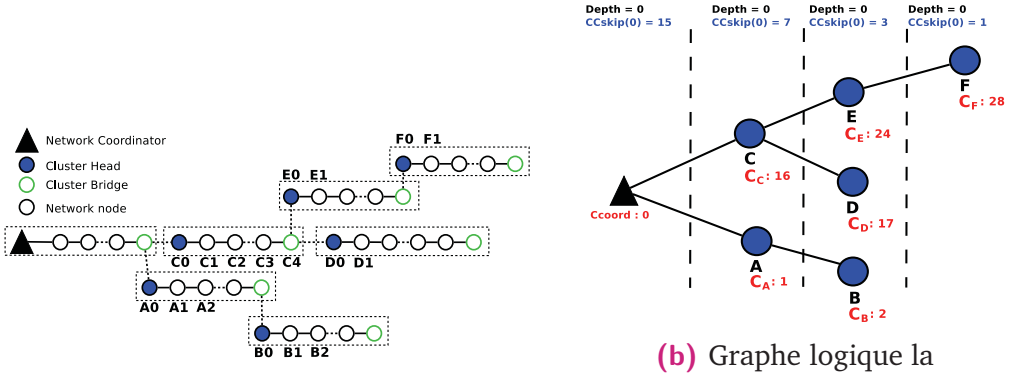
Le nœud $32$ ajoute dans sa table, pour son nœud fils $56$, l'adresse du nœud préteur $48$. Le nœud $48$ ajoute dans sa table, pour son adresse prêtée $56$, l'adresse du nœud emprunteur $32$. De ce fait, si le nœud $32$ reçoit un paquet à destination du nœud $56$, il ne sera pas envoyé au nœud $48$ comme le ferait *DAAM* mais plutôt à son nœud fils $56$.

Dans des réseaux denses avec beaucoup d'échecs d'associations qui engendrent des emprunts d'adresses, *DIBA* peut entraîner une congestion du réseau due aux nombreuses échanges de messages et aussi des tables de routage complexes. Ce phénomène est exacerbé en cas d'ajout d'une branche entière car dans le cas des réseaux linéaires le mécanisme ne prévoit que des emprunts d'une adresse à la fois. L'autre inconvénient de *DIBA* est qu'un nœud qui obtient une adresse empruntée ne peut plus prêter une adresse parmi son bloc.

### 1.5.3 Mécanisme d'adressage basé sur ZigBee pour réseaux de capteurs sans fils linéaires

Avec le type de topologie en clusters pour les RdCSF linéaires décrit dans la partie 1.4.1, les auteurs de [Pan+08] proposent un mécanisme d'adressage basé sur celui DAAM de ZigBee. Ils proposent de diviser l'adresse 16-bits ZigBee en deux parties. Les $m$ premiers bits de poids forts seront utilisés pour l'adressage des clusters notée $C$ et les $16 - m$ bits de poids faibles pour l'adressage des nœuds dans chaque cluster notée $N$. Par conséquent, un nœud $i$ quelconque du réseau est représenté par l'adresse logique ($C_i$, $N_i$). La distribution des adresses passe d'abord par une phase de planification et de modélisation où il faut définir *manuellement* les emplacements des chefs de cluster (CC) et des ponts de cluster (PC) dans le réseau linéaire. Après cette étape de placement des nœuds, un graphe logique du réseau linéaire est généré (voir la figure 1.30a) où chaque cluster délimité par un chef de cluster et un pont de cluster est transformé en un nœud logique *GL* et où les relations entre les clusters sont transformées en des liens. À partir du graphe GL, les paramètres du réseau sont déterminés, à savoir le nombre maximum de clusters fils par cluster appelé $CC_m$ et la profondeur maximum du réseau de clusters $CL_m$.

La figure 1.30 montre un exemple de planification et de modélisation du graphe logique du réseau de clusters linéaire avant le déploiement des nœuds.




**(a)** Attribution des rôles

**(b)** Graphe logique la figure 1.30a

**Fig. 1.30.:** Adressage d'un arbre de cluster de réseau linéaire

Les auteurs utilisent une approche similaire à *DAAM* de ZigBee pour l'adressage des clusters, c'est à dire des nœuds logiques dans le graphe GL (voir Figure 1.30b). Si $CC_m = 1$, alors les adresses des clusters sont attribuées

de façon incrémentale à partir du cluster du puits. Si $CC_m \geq 2$, alors les adresses des clusters sont distribuées de façon récurrente en fonction de $CCskip(d)$ (voir équation 1.16). Chaque cluster parent ($C_{Parent}$) à la profondeur $d$ dans *GL* attribue un identifiant à chacun de ses clusters fils $C_{fils}$ grâce à l'équation 1.15.

Si $CCm \geq 2$

$$C_{fils} = C_{Parent} + (n-1) \times CCskip(d) + 1 \tag{1.15}$$

$$CCskip(d) = \frac{1 - CCm^{CLm-d}}{1 - CCm} \tag{1.16}$$

Après la phase d'attribution des identifiants de clusters et le déploiement des nœuds, chaque nœud démarre un phase de découverte de voisinage en diffusant un message *HELLO*. À la fin de la phase de découverte de voisinage, le nœud puits initie la diffusion d'une balise contenant son identifiant de cluster $C_0 = 0$ et son identifiant de nœud $N_0 = 0$. Lorsqu'un nœud $v$ avec l'adresse $(C_v, N_v)$ transmet une balise, alors chaque nœud $u$ du même cluster, recevant la balise et n'ayant pas encore obtenue d'identification de nœud $N_u$ , répond par un message *Association_Request*. Le nœud $v$ émetteur de la balise collecte tous les messages*Association_Request*, les trie dans l'ordre croissant de la valeur du *RSSI* reçus et initialise un paramètre $N = N_v + 1$. Puis, pour chaque nœud $u$ émetteur d'une *Association_Request*, le nœud $v$ répond par un message *Association_Response* incluant la valeur $N_u = N$ avant d'incrémenter N ($N = N + 1$). Enfin, le nœud $v$ choisit le dernier nœud $u$ trié pour lui déléguer le rôle de propagation de la balise (*next_beacon_sender(u)*).

Lorsque le nœud $v$ émetteur de la balise est un pont de cluster, il accepte seulement les messages *Association_Request* venant du chef de cluster du cluster fils. Dans ce cas, $v$ répond par un message *Association_Response* avec $N_u = 0$ et délègue au chef de cluster du cluster fils le rôle de propagation de la balise (*next_beacon_sender(u)*).

Le routage des données dans ce réseau se base sur l'organisation hiérarchique des clusters et sur la table de voisinage de chaque nœud obtenue durant la phase d'attribution des adresses. Lorsqu'un nœud $v$ donné reçoit un paquet de données avec comme adresse de destination ($C_{dest}$,$N_{dest}$), si le nœud $v$ est la destination alors il accepte et transmet le paquet à la couche supérieure. Dans le cas où l'identifiant de cluster $C_v$ du nœud $v$ est identique à celui du paquet ($C_{dest}$) , alors le nœud $v$ envoie le paquet de donnée à un nœud $u$ de sa table de voisinage tel que $|N_u - N_{dest}|$ soit la plus petite possible. Si maintenant le cluster du nœud de destination est un descendant du cluster de $v$, c'est à dire un nœud respectant la condition 1.17 alors $v$ choisit, dans sa table de voisinage, un prochain saut $u$ qui remplit la condition 1.18.

$$C_v < C_{dest} < C_v + (CC_m - 1) \times CC_{skip}(d) \tag{1.17}$$

$$C_u < C_{dest} < C_u + (CC_m - 1) \times CC_{skip}(d+1) \tag{1.18}$$

Si ce nœud n'existe pas dans sa table de voisinage, alors il choisit comme prochain saut le nœud avec la plus grande adresse $N_u$.

Dans tous les autres cas, le paquet de donnée est envoyé à un nœud $u$ de sa table de voisinage appartenant au cluster parent qui remplit la condition 1.19.

$$C_u < C_{dest} < C_u + (CC_m - 1) \times CC_{skip}(d-1) \tag{1.19}$$

Si ce nœud n'existe pas alors le paquet est transmis à un nœud $u$ ayant la plus petite adresse dans la table de voisinage.

| **Paramètres de l'arbre de cluster** | | | **Espace d'adresses disponibles** | |
|---|---|---|---|---|
| CCm | CLm | Nbre de bits cluster (m) | Clusters adressables | Nœuds adressables |
| 2 | 11 | 12 | 4094 | 16 |
| 2 | 7 | 8 | 254 | 256 |
| 2 | 5 | 6 | 62 | 1024 |
| 2 | 3 | 4 | 14 | 4096 |

**Tab. 1.4.:** Différentes configurations en fonction du paramètre $m$

Même si le mécanisme permet d'adresser beaucoup plus de nœuds que ZigBee, il souffre de quelques manquements. La phase de sélection et de paramétrage des identifiants de clusters sur chaque nœud se déroule manuellement avant leur déploiement et cette tâche devient fastidieuse face à de

grands réseaux. Le calcul des ID de chaque cluster se base sur la même approche de ZigBee, de ce fait le nombre maximum de clusters qui peuvent être adressés est limité par la taille d'adressage $2^m$ avec $m$ la partie allouée à l'ID cluster dans l'espace d'adressage 16-bits. De plus, la profondeur du réseau de clusters sera fortement impactée par le choix de ce paramètre $m$. Le tableau 1.4 montre les différentes configurations avec pour chacune le nombre de clusters et de nœuds par cluster.

### 1.5.4 Mécanisme d'adressage de routage pour les RdCSFL multi-niveau

Dans [Jaw+09 ; Jaw+11], les auteurs proposent une architecture réseau et un mécanisme d'adressage et de routage pour les RdCSFL. L'architecture proposée définit trois types de nœuds à savoir :

— Les NCS (Nœuds de Capteurs Simples) pour la capture des données locales

— Les NRD (Nœuds de Relais de Données) qui assurent la collecte des données transmises par les NCS situés dans leur voisinage à un saut.

— Les NAD (Nœuds d'Acheminement de Données) qui reçoivent les données des NRD afin de les transmettre au puits ou Centre de Contrôle et de Traitement des données (CCTD).

Un schéma de cette architecture est décrit dans la figure 1.17 de la section 1.4.1. Afin d'assurer l'acheminement et le routage des données capturées vers le puits, un mécanisme d'adressage hiérarchique des différents nœuds au format décimal pointé $NAD.NRD.NCS$ est proposé. Dans ce format d'adressage, la valeur $NAD$ représente l'identification des nœuds NAD, la valeur $NRD$ permet d'identifier les nœuds NRD et celle de $NCS$, les nœuds NCS. Chaque champ est numéroté de 0 à $NT - 1$ où $NT$ représente le nombre total du type de nœuds correspondant au champs à identifier.

Lorsqu'il s'agit d'adresser un nœud NCS, le dernier champ $NCS$ représente l'identifiant du nœud lui même, le deuxième champ $NRD$ représente l'identification du nœud NRD le plus proche qui collecte les données du nœud NCS et le champ $NAD$ représente l'identification du nœud NAD qui collecte les données du nœud NRD du champ $NRD$. Pour adresser un nœud NRD, le

champ $NCS$ est égal au nombre total de nœuds NCS, le champ $NRD$ représente son identifiant et le champ $NAD$ le nœud NAD parent à qui il envoie ces données collectées. Pour un nœud NAD, les champs $NRD$ et $NCS$ ont comme valeurs respectives le nombre total de nœuds NRD et le nombre total de nœuds NCS et le champ $NAD$ représente l'identifiant du nœud NAD lui même.

Par exemple, pour un réseau RdCSFL composé de 3 nœuds NAD, de 2 nœuds NRD par NAD et de 4 nœuds NCS par NRD, les nœuds NAD seront numérotés de 0 à 2, chaque groupe de nœuds NRD sera numéroté de 0 à 1. Et pour les groupes de NCS, ils seront identifiés chacun de 0 à 3.

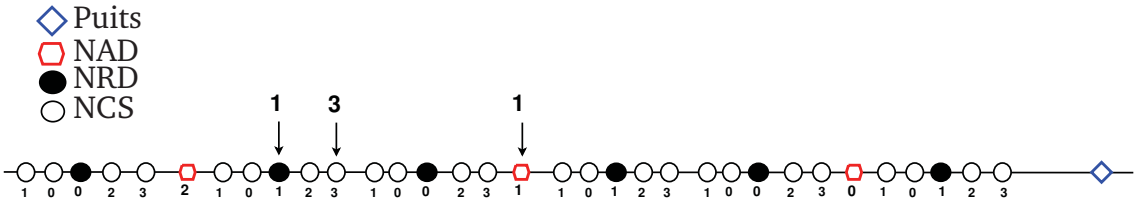



**Fig. 1.31.:** Adressage d'un réseau linéaire à 3-niveaux

La figure 1.31 montre une adressage hiérarchique au format décimal pointé. Le nœud ayant comme adresse 1.1.3 représente le nœud NCS 3 du nœud NRD parent 1 du nœud NAD 1.

La collecte des données par les nœuds NRD se fait par simple transmission à 1-saut car tous les nœuds NCS sont à portée de leur NRD parent respectif. Les nœuds NRD quant à eux transmettent leurs données collectées en utilisant un protocole de routage multi-saut ZigBee ou Jump Always et Redirect Always [Jaw+09].

## 1.6 L'approche multi-puits dans les réseaux de capteurs sans-fil

L'une des problématiques majeures dans les réseaux de capteurs sans-fil est le drainage des données jusqu'au puits. En effet, lors de l'acheminement des données vers un seul puits, la totalité du trafic venant des nœuds sources est relayée par les nœuds au voisinage du puits. Ainsi pour compenser les dé-

fauts bien connus du routage vers un seul puits dans les RdCSF et dans les RdCSF linéaires en particulier (encombrement, débordement de files d'attente, épuisement précoce de l'énergie des nœuds au voisinage du puits), des puits secondaires peuvent être disséminés dans le réseau.

Des mécanismes de routage multi-puits sont proposés dans la littérature pour les réseaux de capteurs sans-fil de façon générale [MP11 ; Mig+15 ; Moo+10] pour résoudre les problèmes de surcharge du réseau, d'épuisement de l'énergie des nœuds au voisinage du puits, mais aussi pour minimiser les nombres de retransmissions et de pertes de paquets. Dans les sections suivantes, nous présentons quelques uns de ces mécanismes de routage multi-puits.

### 1.6.1 MUltiSource MUltiSink Trees for Energy-efficient Routing (MUSTER)

Les auteurs dans [MP11] ont proposé un protocole de routage multi-puits afin de minimiser le nombre de liens relais de transmission utilisés pour le routage des données venant de plusieurs nœuds de capteurs sources à destination de plusieurs puits. *MUSTER* construit des arbres dont les racines sont les nœuds puits et qui sont composés de groupes de nœuds sources, et de nœuds relais en utilisant un mécanisme d'inondation et de chemin inverse. Chaque nœud choisit dans son voisinage à 1-saut le prochain nœud relais pour atteindre le ou les nœuds puits. Le choix des nœuds relais comme prochain saut est basé sur la valeur $Q(n, s)$ qui représente la qualité d'un nœud $n$ à être un relais vers le puits $s$.

L'équation (1.20) montre la méthode de calcul de $Q(n, s)$ où $R(n, s)$ représente la qualité de routage du nœud $n$ et T(n), sa durée de vie estimative.

$$Q(n, s) = R(n, s) \times T(n) \tag{1.20}$$

La qualité de routage $R(n)$ (voir équation (1.21)) d'un nœud $n$ est obtenue grâce aux métriques suivantes :

— $reliability(n, s)$ mesure la fiabilité d'un nœud $n$ à effectuer des communications de bout-en-bout jusqu'au puits $s$ ;

— $path(n)$ indique le nombre de chemins source-puits passant par le nœud $n$ ;

— $sink(n)$, indique le nombre de puits à destination de laquelle le nœud $n$ a déjà relayé des paquets.

$$R(n,s) = \alpha \times reliability(n,s) + \beta \times paths(n) + \gamma \times sinks(n) \qquad (1.21)$$

$\alpha$, $\beta$ et $\gamma$ représente des paramètres de réglages pour donner des poids à chaque métrique de l'équation 1.21. Par défaut, les auteurs ont défini les mêmes poids à chaque métrique avec $\alpha = \beta = \gamma = 1$.

*MUSTER* construit des chemins vers les puits qui peuvent se chevaucher. Ainsi pour éviter l'épuisement précoce des nœuds relais les plus sollicités et ainsi réduire la durée de vie du réseau, *MUSTER* réalise un basculement des chemins pour acheminer les données jusqu'aux puits en se basant sur les valeurs de l'équation (1.20).

Dans l'exemple de la figure 1.35, le protocole commence à construire les arbres de routage permettant d'atteindre les différents puits (figure 1.32). L'exemple de la figure 1.33 montre un chevauchement des chemins pour une optimisation de la consommation de l'énergie au niveau des nœuds relais. Dans la figure 1.34, *MUSTER* procède à un basculement des chemins vers d'autres nœuds relais afin de répartir la consommation de l'énergie et assurer une plus longue durée de vie du réseau.

Dans le cas d'un RdCSF linéaire où le trafic de chaque nœud est relayé par le même prochain saut vers une destination donnée, MUSTER n'est pas adapté car le mécanisme de basculement et de choix des routes proposés ne sert à rien puisque tout le trafic d'un nœud passe par un même prochain saut pour atteindre la destination

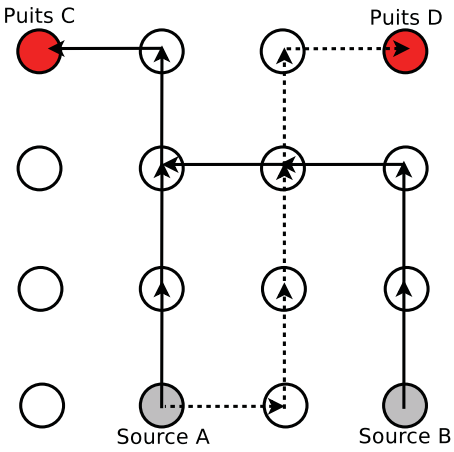



**Fig. 1.32.:** Exemple de scénario multi-source vers multi-puits

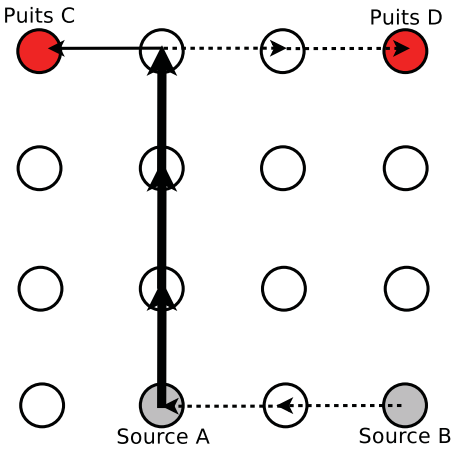



**Fig. 1.33.:** Exemple de scénario avec chevauchement des chemins

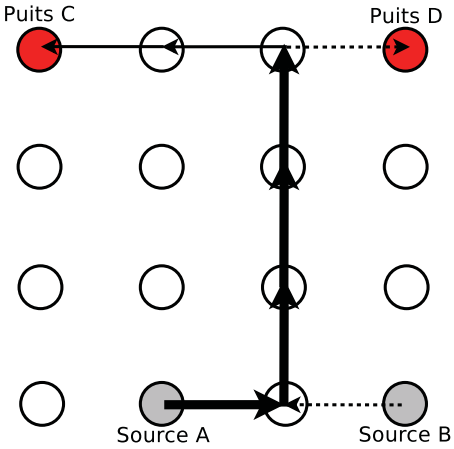



**Fig. 1.34.:** Basculement des chemins sur d'autres chemins

Chemin agrégé ← Arbre routé en C ← Arbre routé en D ←

**Fig. 1.35.:** Exemple de Routage avec MUSTER

### 1.6.2 Generic energy-Balancing Gradient-Based Routing protocol (GB-GBR)

Dans [Mig+15], les auteurs proposent un protocole qui est une variante du protocole *Direct Diffusion* [Int+00] qui permet de maximiser la durée de vie des nœuds du réseau en évitant les liens les plus surchargés durant le relayage des paquets de données. Les puits diffusent les requêtes dans le réseau à destination des nœuds sources de données. Chaque nœud relais diffuse à nouveau la requête en prenant le soin de récupérer les informations de son nœud voisin prochain saut qui lui a transmis la requête et qui lui permettra de joindre le puits. Ces informations incluent le nombre de sauts vers le puits, l'énergie résiduelle, l'adresse du nœud. Le nœud source, pour envoyer la réponse au puits à travers une route $k$, choisit le prochain nœud relais en fonction du nombre de sauts pour atteindre le puits $Nhops_k$, de la moyenne d'énergie restante $AvgRE_k$ sur les nœuds relais du chemin $k$, et de la consommation d'énergie estimée $\Delta En$ sur le chemin $k$. Chaque nœud choisit, dans son voisinage, le prochain nœud relais ayant la plus grande valeur de la fonction (1.22)

$$Max\{AvgRE_k - Nhops_k \times \Delta En\} \tag{1.22}$$

Cependant pour un RdCSF linéaire, l'utilisation de *GB-GBR* pour le calcul du meilleur chemin vers un puits donné n'est pas très approprié. En effet, l'une des caractéristiques d'un RdCSF linéaire est que chaque nœud transmet ses propres données et relaie les données venant de son prédécesseur vers une destination donnée. De ce fait, le débit de trafic acheminé par un nœud augmente au fur et à mesure qu'on se rapproche du puits occasionnant ainsi un épuisement précoce des nœuds au voisinage du puits. De plus, l'utilisation de la moyenne de l'énergie restante $AvgRE_k$ dans le calcul du meilleur chemin vers une destination ne permet pas de déceler et de désavantager les nœuds critiques qui ont peu d'énergie résiduelle. En effet, un chemin vers un puits peut comporter un ou plusieurs nœuds critiques (proches du puits) et des nœuds avec de très grandes énergie résiduelles (très éloignés du puits) et que la moyenne donne une valeur qui fait que ce chemin soit meilleur par rapport à un autre chemin sans nœud critique.

### 1.6.3 Gradient-based routing protocol for LOad-BALancing (GLOBAL)

Dans [Moo+10], les auteurs proposent aussi un protocole permettant de maximiser la durée de vie du RdCSF en évitant les liens les plus surchargés durant le relayage des données vers les nœuds puits. Chaque nœud du réseau calcule le gradient vers chaque puits en se basant sur le nombre de sauts vers les puits et sur le ratio d'épuisement de l'énergie résiduelle "moyenne" par unité de temps (voir équation (1.23)) de chaque nœud relais *REDR* (Residual Energy Depletion Rate) sur le chemin vers les puits.

$$REDR_{node(i)} = \alpha \times REDR_{old} + (1-\alpha) \times REDR_{sample} \quad (1.23)$$

$$\text{Où } \alpha = 0.3$$

$$REDR_{sample} = \frac{(1-\frac{re_{curr}}{re_{curr-T}})}{T} \quad (1.24)$$

$re_{curr}$ représente l'énergie résiduelle courant à l'instant $curr$.

Pour sa mise en œuvre, *GLOBAL* utilise des messages d'annonces *ADV* qui inondent le réseau à partir des nœuds puits. Les messages *ADV* contiennent le nombre de sauts $hcnt$ traversés depuis le puits, la somme cumulée des *REDR* $sum_{redr}$ des nœuds traversés depuis le puits et le maximum des valeurs *REDR* $max_{redr}$ des nœuds traversés.

$$G_{node(i)} = \beta \times sum\_redr_L + (1-\beta) \times MAX(max\_redr, REDR_{node(i)}) \quad (1.25)$$

À la réception d'un message *ADV*, chaque nœud $n$ calcule le nouveau gradient $Gn$ grâce à l'équation (1.25). Si le gradient $Gn$ est plus petit que le $Gncurr$ et que le nombre de sauts vers le puits $hcnt + 1$ est inférieur à $K$, représentant le nombre de sauts maximum, alors le nœud $n$ choisit ce nœud voisin comme prochain relais "courant" vers le puits. Ensuite, le nœud $n$ fait la mise à jour des valeurs $hcnt$, $sum_{redr}$ et $max_{redr}$ du message *ADV* avant de le rediffuser dans le voisinage.

Dans les RdCSFL, l'utilisation des techniques d'inondation par les protocoles de routage multi-puits tel que GLOBAL n'est pas très adaptée. En effet, tous

les messages *ADV* provenant d'un puits à destination d'un nœud du réseau suivent le même chemin. De plus comme dans GB-GBR, la technique de calcul du gradient basée sur la somme cumulée de l'énergie consommée sur le chemin vers le puits ne permet ici d'écarter systématiquement les chemins contenant des nœuds critiques ayant une énergie résiduelle très faible.

## 1.7 Conclusion

Dans cette partie, nous avons présenté des standards, protocoles ainsi que des mécanismes utilisés dans les différentes couches protocolaires pour les réseaux de capteurs sans-fil en général et les RdCSF linéaires en particulier. Cet état de l'art a permis d'aborder les standards LR-WPAN et LPWAN, les protocoles MAC et les types de topologies dans les réseaux de capteurs sans fil et les mécanismes de contrôle de topologies et aussi de mettre en exergue l'inexistence de mécanismes permettant de construire des topologies logiques adaptées aux RdCSF linéaires.

Les protocoles MAC adaptés pour les réseaux de capteurs sans fils linéaires que nous avons étudiés ne prennent en charge que des réseaux composés d'une seule ligne de nœuds capteurs. Dans le cas d'un réseau avec plusieurs branches, ces protocoles ne gèrent pas, au voisinage d'une zone de branchement, la gestion des cellules et des réservations [Wat+06], planification des périodes de réveil et de relayage [Fan+11], et de gestion des *Token* [Ndo+16] venant de chaque extrémité d'une branche.

Nous avons aussi fait l'état de l'art des mécanismes d'adressage utilisés pour le réseau en arbre ZigBee et les RdCSF linéaires. Ces techniques d'adressage présentées dans cette partie souffrent pour la plupart d'une manque de flexibilité surtout pour un RdCSF linéaire dont on ne connaît pas à l'avance les caractéristiques telles que le nombre de branchements de ce réseau ainsi que la profondeur. Par exemple, ZigBee est plus adapté pour un réseau d'arbres de clusters équilibré et où la profondeur ne dépasse pas les $15$ nœuds avec $C_m = 2, Rm = 2$ afin d'éviter les problèmes d'apparition de nœuds orphelins ou de gaspillage de blocs d'adresses. Dans [Pan+08],le nombre de clusters possibles, de nœuds par cluster et la profondeur du réseau de clusters dans un tel réseau linéaire dépendent fortement du choix du paramètre $m$ délimi-

tant les deux parties du champs d'adressage (ID du cluster et ID du nœuds). À cela s'ajoute le fait que les nœuds de connexion des clusters (Pont et chef de cluster) sont choisis manuellement. Cette solution statique peut occasionner un partitionnement du réseau en cas d'épuisement de ces nœuds. Enfin le dernier inconvénient de cette proposition de [Pan+08] est le fait que les identifiants de cluster pour chaque nœud sont paramétrés manuellement avant le déploiement du réseau ce qui n'est pas très efficace pour un grand réseau mais aussi pour un déploiement rapide en environnement hostile. Le mécanisme proposé par les auteurs [Jaw+09] ne permet pas une auto-organisation car le nombre de NAD, de NARD et de NCS sont définis avant le déploiement du réseau ce qui ne permet pas une extension du réseau avec l'ajout de nœuds lorsque qu'une nouvelle branche d'un RdCSF linéaire doit par exemple être surveillée. De plus, le manque de redondance des NRD peut occasionner un partitionnement du réseau en cas de perte d'un nœud NRD.

Nous avons terminé notre état de l'art, avec la description des mécanismes de routage multi-puits. Dans ces différents mécanismes de routage, l'approche de calcul du meilleur chemin (utilisation de la somme cumulée de l'énergie consommée ou du la moyenne de l'énergie restante sur un chemin donné) ne permet pas d'éviter les nœuds critiques (ayant une forte consommation d'énergie ou disposant de peu d'énergie résiduelle) sur un chemin vers un puits.

Dans les deux chapitres suivants, nous présentons nos différentes contributions de mécanismes de formation de topologies linéaires, d'adressage et de gestion dynamique des nœuds adaptés aux RdCSF linéaires ainsi que les résultats de leur évaluation sur l'environnement de simulation Castalia/Omnet++.

# 2

# Mécanisme de formation des RdCSF Linéaires

Dans les réseaux de capteurs sans-fils linéaires, les nœuds de capteurs sont déployés le long de plusieurs lignes reliées au niveau des zones de jonctions appelées branchements et ceci sur de très longues distances. L'étude de l'état de l'art nous a permis de montrer que les mécanismes de formation et d'adressage proposés pour les RdCSF en général ne sont pas adaptés au cas des RdCSFL. De plus, les quelques mécanismes existants pour les RdCSFL ne permettent pas une bonne organisation et une formation en toute autonomie des nœuds de capteurs, ne minimisent pas les interventions et configurations manuelles au moment du déploiement des nœuds de capteurs et n'optimisent pas la répartition des adresses.

Pour cette partie de notre contribution, nous étudions des RdCSFL composés de nœuds de capteurs ayant les fonctionnalités de capture de données locales et de routeur. Dans ces réseaux nous avons trois types de trafics :

— des trafics de données descendants : émis par le puits (aussi appelé point de collecte ou PAN coordinateur selon la solution décrite dans le Chapitre 1) à destination des nœuds du réseau ;

— des trafics de données montants : émis par les nœuds de capteurs à destination du puits ;

— des trafics de contrôle : émis par chaque nœud à destination d'autres nœuds du réseau.

Dans notre premier contribution, nous proposons un mécanisme qui permet la formation et la découverte de topologie de réseaux de capteurs sans-fil linéaires appelé *DiscoProto* sans connaissance au préalable des caractéristiques du réseau à savoir la profondeur du réseau, le nombre maximum de nœuds fils par nœud et sans aucune intervention manuelle hormis le choix du puits. Les nœuds du RdCSFL n'ont que la connaissance de leur voisinage qu'ils découvrent par échange de messages. Le protocole *DiscoProto* a pour but de contrôler la formation de la topologie afin d'éviter au maximum la formation en étoile des nœuds et de fournir une topologie logique le plus proche possible de la topologie physique des nœuds. Nous appelons

topologie physique la configuration spatiale des nœuds de capteurs correspondant à l'environnement de déploiement. La topologie logique définit la structure du réseau formée après l'association des nœuds. *DiscoProto* fournit un mécanisme d'adressage hiérarchique initial des nœuds du RdCSFL. Notre algorithme s'inspire de l'algorithme de Prim (algorithme de Jarnik) [Jar30 ; Pri57] et en propose une version distribuée. Il permet d'étendre à chaque itération la construction d'un arbre linéaire recouvrant de poids maximum dont la racine est le puits du RdCSFL.

## 2.1 Les réseaux de capteurs sans-fil linéaires

### 2.1.1 Les Réseaux de capteurs sans-fil strictement linéaires

Dans les RdCSFL, la plupart des mécanismes et solutions proposés traitent le cas des architectures strictement linéaires c'est à dire sur des segments de droites où les nœuds de capteurs sont alignés le long de ces segments linéaires [Jaw+11 ; Ndo+14a]. Dans ces travaux, les auteurs définissent un RdCSFL comme étant un réseau où chaque nœud a, dans la zone de couverture du signal radio, au plus $2 * K$ voisins (K nœuds dans chaque direction). Les auteurs dans [Ndo+14a] ont défini $K$ comme étant le facteur de redondance des nœuds de capteurs dans le RdCSFL. Cette notion peut être rapprochée de la $K$-connexité des graphes.

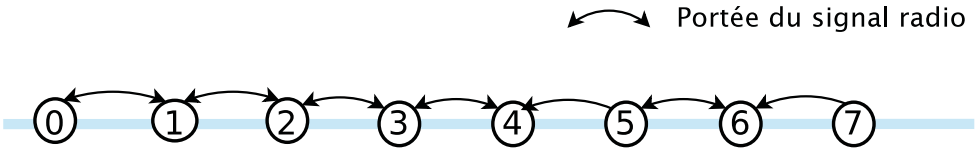



**Fig. 2.1.:** Topologie 1-redondante

L'exemple de la figure 2.1 montre un RdCSFL 1-redondant où chaque nœud de capteurs a au plus deux voisins, un dans chaque direction. L'exemple de la figure 2.2 montre quant à lui un RdCSFL 2-redondant où chaque nœud de capteurs a au plus quatre voisins, deux au plus dans chaque direction (voir figure 2.2).

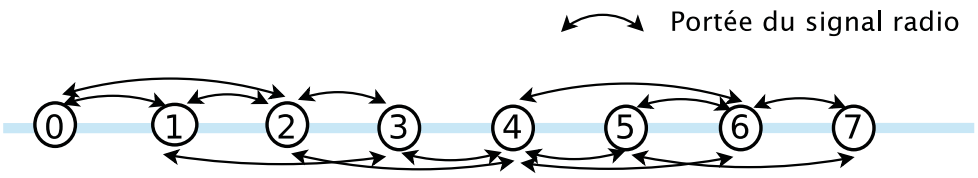



**Fig. 2.2.:** Topologie 2-redondante

Cependant, cette définition strictement linéaire ne s'adapte pas aux RdCSFL de façon générale comme dans les environnements tels que les cours d'eau, les autoroutes, les lignes de chemin de fer où les RdCSFL utilisés sont composés de plusieurs lignes pouvant comporter des zones de jonction ou branchement.

### 2.1.2 Les Réseaux de capteurs sans-fil linéaires avec des branchements

Le déploiement des nœuds de capteurs dans ces types d'environnement de RdCSFL comportant des zones de jonction ou de branchement montre que certains nœuds se trouvant dans le voisinage de ces zones ont un nombre de voisins supérieurs à $2 * K$. La figure 2.3 montre une topologie $2\text{-}redondant$ avec une zone de branchement. Chaque cercle représente la couverture radio du nœud de même couleur. Nous voyons que les nœuds situés dans le voisinage de la zone de branchement ont un nombre de nœuds voisins supérieur à quatre.

De ce fait, il est nécessaire de fournir une nouvelle approche afin de donner une définition beaucoup plus générale pour distinguer les RdCSFL des autres types de réseaux (en étoile, maillé, arbre de clusters).

En effet, dans certains types de configurations, pourtant composées de lignes, les RdCSFL ont une topologie physique résultante qui ressemble plus à une topologie maillée qu'à une topologie linéaire. La figure 2.4a montre un exemple de topologie où les nœuds de capteurs sont déployés le long des lignes avec des branchements. Lorsque cette topologie est visualisée sans les lignes (figure 2.4b), elle ressemble plus à une topologie maillée qu'à une topologie linéaire.

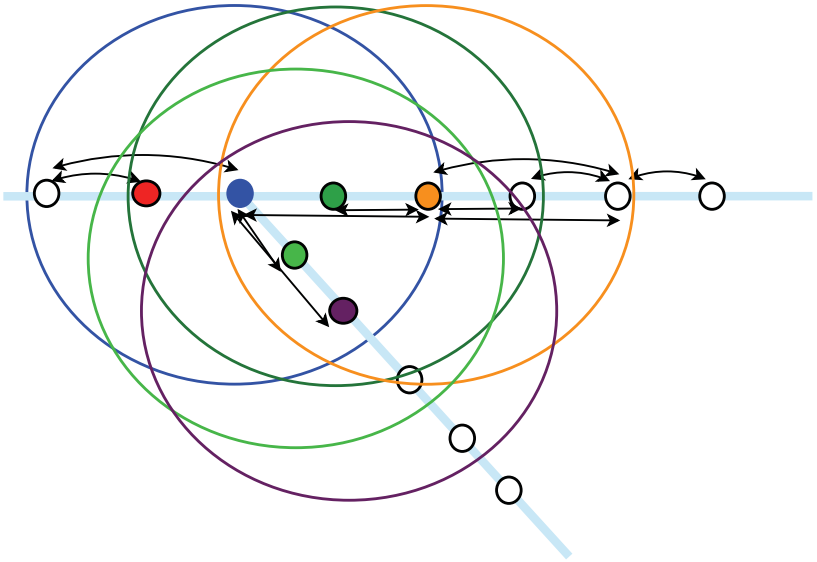
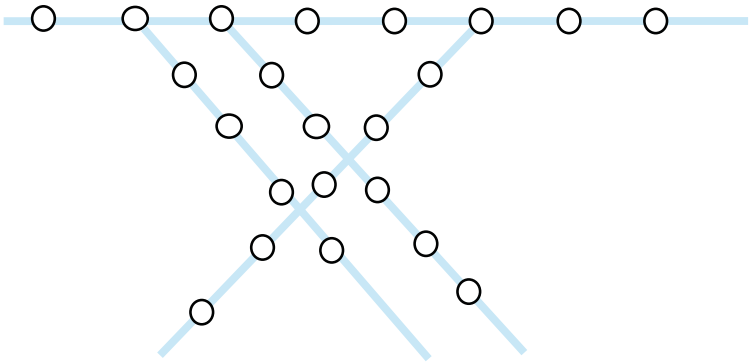
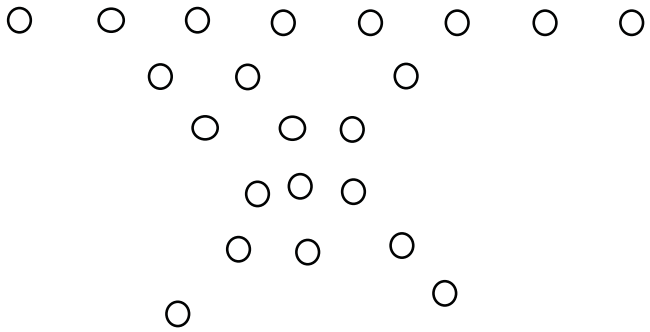


**Fig. 2.3.:** Topologie 2-redondante composée de deux lignes avec une zone de jonction



**(a)** Topologie avec les lignes de construction rappprochées

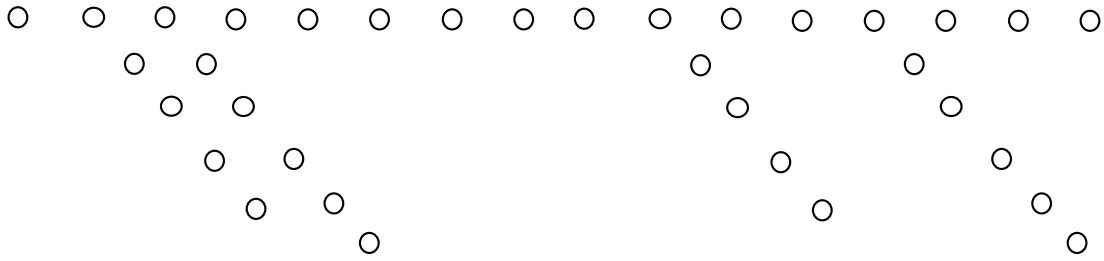


**(b)** Est-elle linéaire ?

**Fig. 2.4.:** Exemple de RdCSFL avec branchements

### 2.1.3 Facteur de linéarité

Pour définir un RdCSFL, nous introduisons une métrique appelée **facteur de linéarité** *lF (linearity Factor)* (linearity factor) qui représente le pourcentage de nœuds simples dans le réseau. En effet, nous définissons deux types de nœuds dans un RdCSF linéaire $K\text{-}redondant$. Les nœuds de capteurs avec au plus $2 * K$ appelés *nœuds simples* et les nœuds situés dans le voisinage des zones de branchement ayant au moins $2 * K + 1$ voisins appelés *nœuds de jonction* ou *nœuds de branchement*.



**(a)** Facteur de linéarité de $29.4\%$ **(b)** Facteur de linéarité de $41\%$

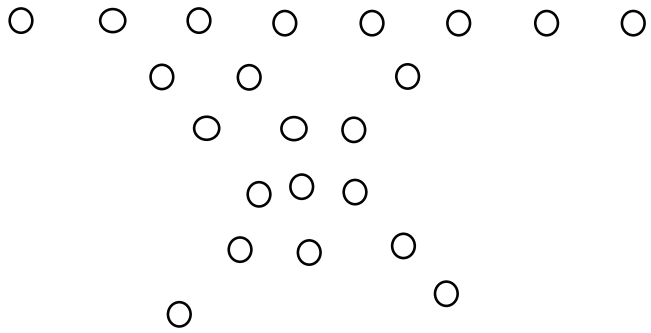
**Fig. 2.5.:** Facteur de linéarité d'un exemple RdCSFL composé de $3$ lignes

L'estimation de ce facteur de linéarité va nous permettre de considérer n'importe quel réseau de capteurs comme linéaire ou pas. De ce fait, pour chaque type d'environnement (d'application), nous pouvons définir une valeur *seuil de linéarité* et un RdCSF sera considéré linéaire si son facteur de linéarité est supérieur ou égal à la valeur seuil définie.

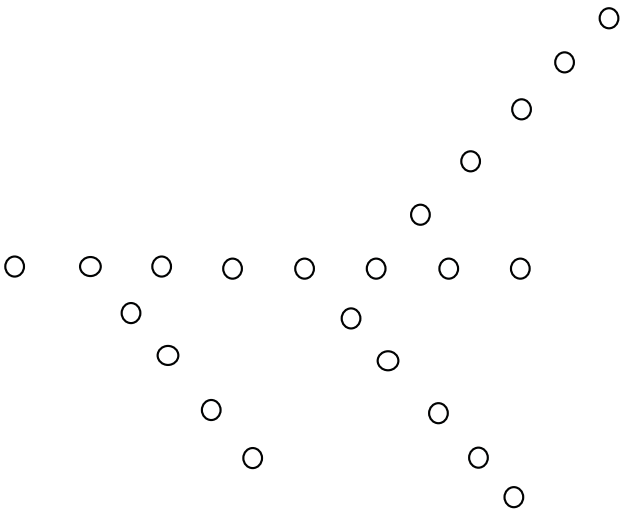
Les figures 2.5 et 2.6 montrent respectivement deux topologies avec des configurations différentes. Suivant les localisations des zones de branchement, ils fournissent des facteurs de linéarité différents.

Dans la suite de nos travaux nous adoptons la définition suivant pour les RdCSFL.

**Définition d'un RdCSFL** :
Soient **N**, l'ensemble des nœuds d'un réseau, $K$ le facteur de redondance et $lF$ le facteur de linéarité.
Nous considérons un RdCSFL R(N,K,lF) comme étant un réseau linéaire où le facteur de linéarité $lF$ est supérieur au seuil de linéarité.



**(a)** Facteur de linéarité de $19\%$



**(b)** Facteur de linéarité de $45.50\%$

**Fig. 2.6.:** Facteur de linéarité d'un exemple RdCSFL composé de $4$ lignes

Fixer un seuil de linéarité , nous permet d'étudier nos différentes contributions sur les réseaux linaires. Avec un seuil haut, nous allons étudier des réseaux avec peu de branchements et avec un seuil bas nous allons étudier des réseaux avec beaucoup de branchements. Pour notre premier contribution, nous allons étudier le comportement de notre protocole sur tout type de réseaux afin d'évaluer sa robustesse. Dans la partie 2.7.6 de la page 112, nous utilisons des topologies de RdCSFL avec un facteur de linéarité supérieur à $60\%$.

## 2.2 L'algorithme de Prim

L'algorithme de Prim (algorithme de Jarnik) [Jar30 ; Pri57] est un algorithme glouton qui permet de construire un arbre recouvrant minimal, à partir d'un graphe connecté, non orienté et valué.

Soit un graphe $G = (V, E)$ où **V** est l'ensemble des nœuds ou sommets du graphe et **E** l'ensemble des arrêtes ou liens. L'algorithme de Prim décrit dans l'algorithme 2.1, dont le fonctionnement complet et l'implémentation est détaillé dans [KT05], permet construire un arbre recouvrant minimal $T = (V, E')$ avec $E' \subset E$ et dont la racine est un sommet de $V$. Pour ce faire, des poids sont affectés à chaque arrête $e = (v, u)$ du graphe où $e \in E$ et $v, u \in V$. À chaque itération, le nœud ayant le lien de poids minimal avec les nœuds de $T$ est choisi pour être ajouté à l'arbre.

Les exemples de la figure 2.7 à la page 77 montrent des étapes de la construction d'un arbre recouvrant avec l'algorithme de Prim. Considérons l'exemple de la figure 2.7b. Pour construire l'arbre recouvrant minimal $T$ à partir du nœud **a**, l'algorithme de Prim choisit le lien de poids minimal parmi les suivants $(a, b) = 4$, $(a, c) = 3$ et $(a, e) = 7$ (voir Figure 2.7c). Ensuite à partir de l'étape $1$, nous avons deux nœuds $a, c$ qui appartient à l'ensemble $T$. L'algorithme de Prim va ensuite choisir, parmi les liens commençant par **a** et **c**, celui ayant la plus petite valeur c'est à dire le lien $(c, b) = 2$ (voir figure 2.7d). L'algorithme de Prim continue ainsi la construction du graphe jusqu'à inclure tous les nœuds de l'ensemble **V** dans l'arbre $T$ avec les liens de poids minimal (voir figure 2.7j).

**Algorithme 2.1:** Fonctionnement de l'algorithme de Prim

**Result:** Construction de l'arbre recouvrant minimal $T = (V, E')$
**Input :** Graph $G = (V, E)$
**Output :** Arbre $T = (V, E')$

```
1  U ← ∅;
2  U ← G ;
3  T.Ajouter (U.Debut);
4  U.Supprimer (U.Debut);
5  while U ≠ ∅ do
6      min ← CoutMax ;
7      foreach t ∈ T do
8          foreach u ∈ U do
9              if Poids (t,u) < min then
10                 min ← Poids (t,u);
11                 umin ← u ;
12     T.Ajouter (umin);
13     U.Supprimer (umin);
```

Cependant, l'algorithme de Prim est centralisé et pour la construction de l'arbre recouvrant minimal, l'algorithme a besoin de connaître l'intégralité du graphe. Ce qui est difficilement applicable dans le cas des RdCSF car il faudra avoir un nœud capteur capable de centraliser les informations de coût et voisinage de chaque nœud du réseau pour effectuer le calcul de l'arbre recouvrant.

Dans la suite, nous présentons l'approche centralisée et l'approche distribuée de notre protocole *DiscoProto*.

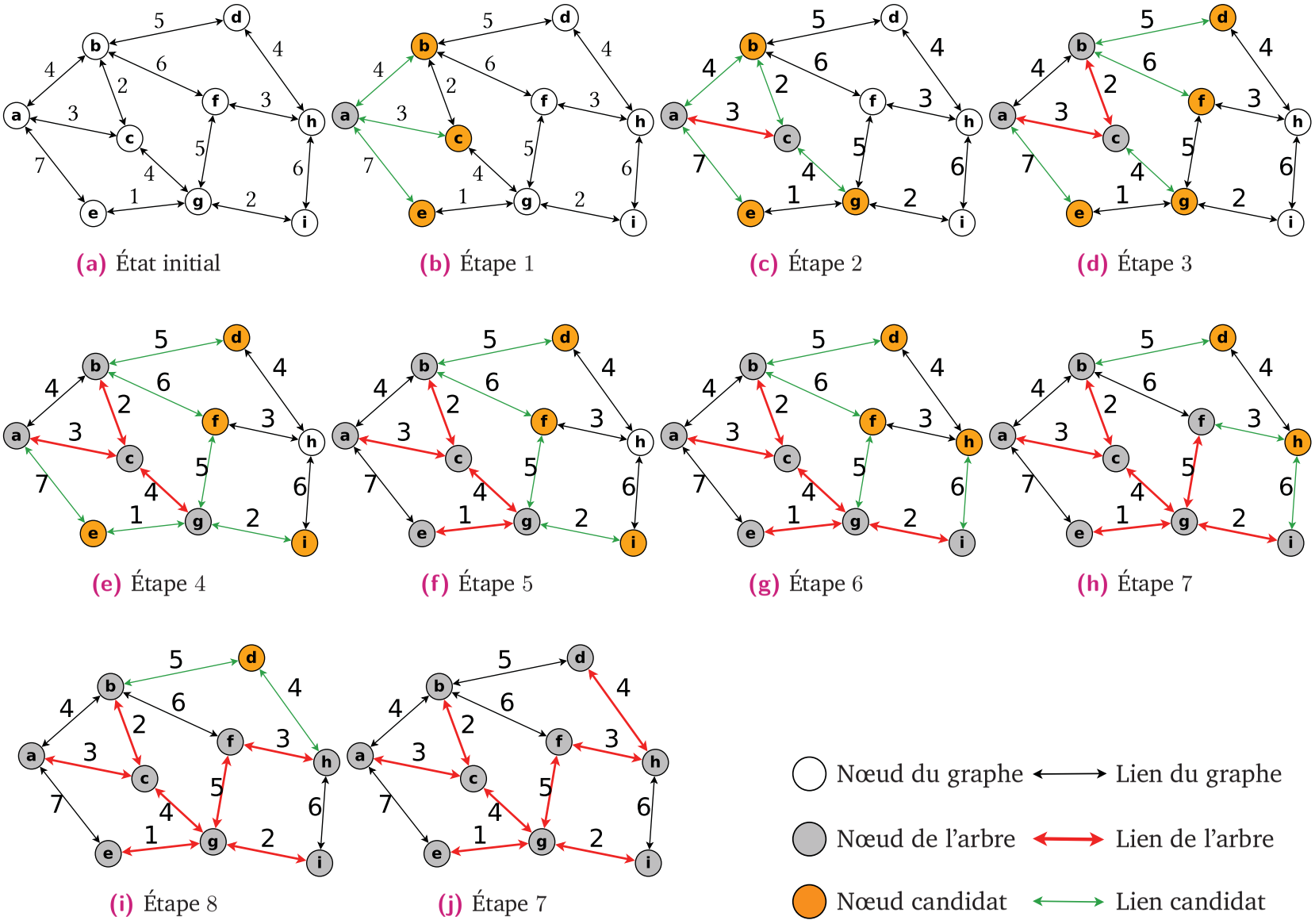



**Fig. 2.7.:** Fonctionnement de l'algorithme de Prim

## 2.3 L'approche centralisée de DiscoProto

Nous considérons un réseau où tous les nœuds ont des fonctionnalités de capteur et de routeur. Dans ce réseau, nous avons deux ensembles **C** et **N** définissant respectivement l'ensemble des nœuds connectés et non connectés. Deux nœuds du réseau sont considérés comme voisins si et seulement si les échanges de trames se font avec un taux de réussite acceptable par rapport au modèle de propagation. Le lien d'association $L(c, n)$ entre deux nœuds $c \in$ **C** et $n \in$ **N** est créé si et seulement si $c$ et $n$ sont dans leur voisinage respectif.

La découverte et la formation du réseau linéaire équivalent à trouver et à construire un arbre recouvrant avec un minimum de branchements possibles grâce à la seule connaissance du voisinage de chaque nœud. Minimiser le nombre de branchements permet d'identifier les disjonctions dans la topologie physique linéaire.

Nous cherchons à calculer un **arbre recouvrant maximum** où le coût (à maximiser), appelé valeur *Objectif*, associé à chaque lien $L(c, n)$ est **local** et **dynamique** (le coût évolue lors de l'exécution de l'algorithme) et est représenté par la notation $Objectif(c, n)$. Le coût des liens entre les nœuds permet de construire un arbre avec le minimum de branchements et ainsi d'identifier les disjonctions dans un RdCSFL.

Ainsi, la version centralisée de notre algorithme *DiscoProto* est un algorithme *glouton* et permet d'étendre à chaque itération l'arbre composé de l'ensemble **C** avec des nœuds de l'ensemble **N**. *DiscoProto* construit progressivement le réseau connecté avec les meilleurs liens d'association disponibles $L(c, n)$.

Le coût des liens d'association ($Objectif$) entre deux nœuds voisins $c$ et $n$ dépend fortement dans ce contexte de la proximité de voisinage de ces nœuds. De ce fait, cette valeur $Objectif(c, n)$, associée à chaque lien $L(c, n)$, qui permet à un nœud $n$ du réseau de choisir avec quel nœud $c$ de l'arbre il va s'associer, est calculée grâce à trois critères principaux :

1. La proximité de voisinage entre ces deux nœud notée par *Commun(c,n)* ;

2. La position du nœud dans le réseau (nœud d'extrémité ou nœud d'intérieur) est déterminée grâce à la somme des nœuds voisins noté *Voisin(n)* ;
3. La linéarité de la formation

Nous allons définir et expliquer le rôle de chacun de ces critères.

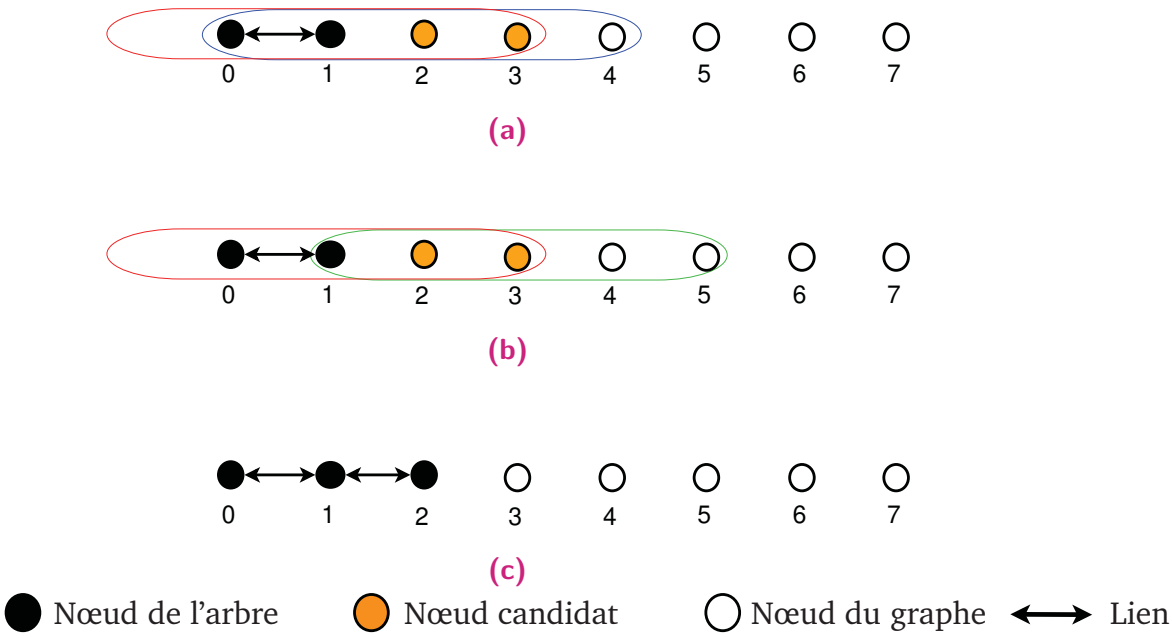
### 2.3.1 Notion de proximité dans les RdCSFL

Nous avons vu dans les parties précédentes que le déploiement des RdCSFL est structuré le long des lignes pouvant comporter des zones de branchement. Suivant le mode de déploiement K-redondant, les nœuds simples ont au plus $2 * K$ voisins et que les nœuds de jonctions ont au moins de $2 * K + 1$ voisins. Durant la construction de l'arbre, un nœud de l'arbre choisit, dans son voisinage de l'ensemble non connecté, le lien d'association avec le nœud le plus proche. Comme les nœuds de capteurs ne sont pas équipés de capteur GPS afin de déterminer leur position et que nous ignorons la topologie physique, nous calculons le nombre de voisins en commun entre deux nœuds afin d'estimer leur proximité .

Considérons une portion d'un réseau $2$-redondant composée de huit nœuds uniformément distribués. Soit $Commun(n, m)$ le nombre de voisin en commun entre deux nœuds n et m. Le nœud $n1$ peut être considéré plus proche de $n$ que le nœud $n2$ lorsque le nombre de voisins en commun noté $Commun(n, n1) > Commun(n, n2)$.

Dans la figure 2.8, chaque cercle représente la zone de couverture radio d'un nœud donné. Dans cette exemple, grâce aux informations de voisinage, le nœud $2$ est considéré plus proche du nœud $1$ que le nœud $3$ car le nœud $1$ a deux voisins en commun avec le nœud $2$ à savoir les nœuds $0$ et $3$ (figure 2.8a) alors qu'il n'en a qu'un avec le nœud $3$ à savoir le nœud $2$ (figure 2.8b). Dans ce cas, le nœud $1$ va choisir d'associer le nœud $2$ (figure 2.8c).

Cependant, cette approche de calcul de la proximité n'est valable que sur le long des parties linéaires d'un réseau. Au niveau des zones de branchement ou lorsque deux branches sont parallèles, les nœuds ont la même valeur nombre de voisins en commun et la seule connaissance de cette métrique




- Voisin(1) = {0,2,3}
- Voisin(2) = {0,1,3,4}
- Voisin(3) = {1,2,4,5}
- Commun(1,2) = CARD{0,3} = 2
- Commun(1,3) = CARD{2} = 1

**Fig. 2.8.:** Notion de proximité dans un RdCSFL

ne permet par de connaître le nœud le plus proche. Pour nous permettre de calculer cette proximité, nous utiliserons en plus du nombre de voisins en commun, la valeur de la moyenne glissante de la puissance du signal reçu (RSSI (Received Signal Strength Indication) des nœuds.

### 2.3.2 Linéarité de la construction de l'arbre

Lors de la construction de l'arbre, il peut arriver que deux nœuds déjà associés à l'arbre soient en compétition pour associer un nœud voisin et que le critère de proximité ne permettent pas de le départager.

Dans l'exemple de la figure 2.9, après les informations de voisinage, le nœud 2 est aussi proche du nœud 4 que le nœud 3 car ils ont le même nombre de voisins en commun (figure 2.9a). Afin de décider qui des deux nœuds 2 et 3 va associer le nœud 4, nous introduisons le critère nombre de nœuds fils qu'un nœud a déjà associés, et à critère nombre de voisin en commun égal, nous privilégions le nœud qui a le plus petit nombre de fils.

Dans ce cas, le nœud 2 a déjà un nœud fils alors que le nœud 3 n'en a pas encore. Donc le nœud 3 sera le nœud qui va associer le nœud 4 (figure 2.9b).

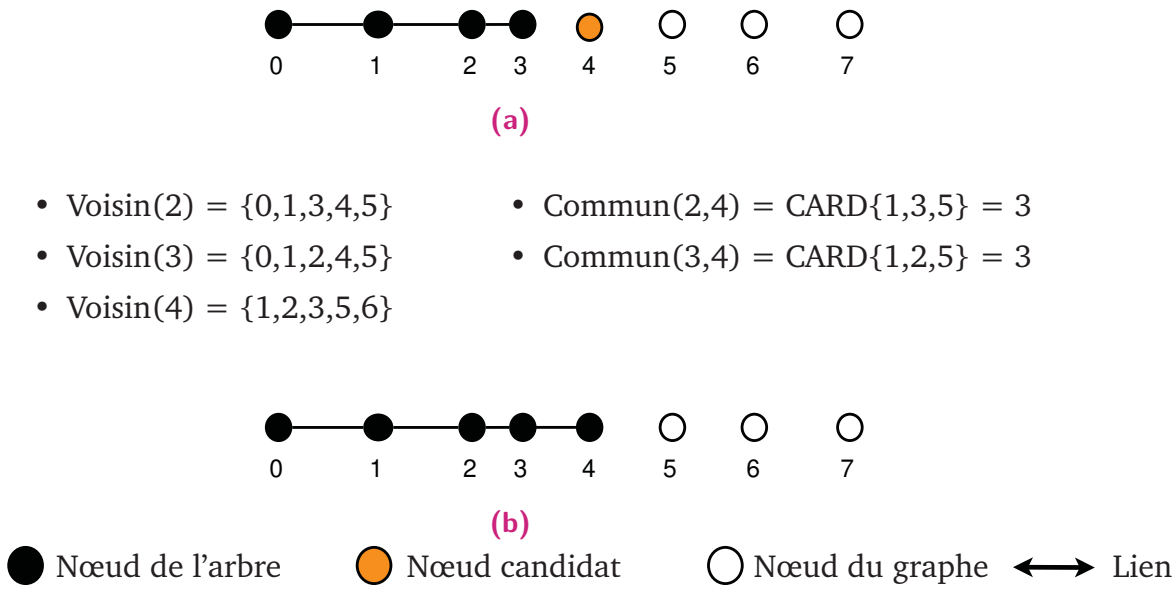
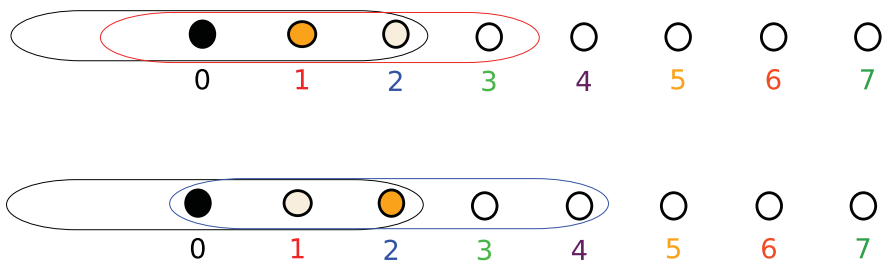



**Fig. 2.9.:** Linéarité dans le choix des nœuds de l'arbre

### 2.3.3 Cas des nœuds d'extrémité

Dans l'évaluation de la proximité des nœuds, il peut exister des cas particuliers où deux nœuds ont les mêmes nombre de voisins en commun. C'est souvent le cas des nœuds situés en extrémité des RdCSFL, qui ont la particularité d'avoir au plus $2 * K - 1$ voisins. En cas d'égalité du nombre de voisins en commun, nous utilisons comme critère secondaire le nombre total de nœuds voisins. L'idée de ce critère est d'éviter les nœuds orphelins en connectant d'abord les nœuds en périphérie, éloignés du réseau.

Dans la figure 2.10, lorsque le nœud $0$ veut déterminer entre les deux nœuds $1$ et $2$ quel est le nœud le plus proche, il remarque qu'ils ont la même valeur de nombre de voisin en commun c'est à dire $Commun(0,1) = Commun(0,2) = 1$. Dans ce cas, grâce au nombre total de nœuds voisins, le nœud $1$ sera considéré comme étant le plus proche du nœud $0$ par rapport au nœud $2$.




- Voisin(0) = {1,2}
- Voisin(1) = {0,2,3}
- Voisin(2) = {0,1,3,4}

- Commun(0,1) = CARD{2} = 1
- Commun(0,2) = CARD{1} = 1

**Fig. 2.10.:** Notion de proximité, Cas des nœuds d'extrémité

En résumé, pour construire la topologie dans un RdCSFL, nous évaluons le coût d'un lien en fonction de ces critères par ordre de priorité.

+ ***Commun(c,n)*** : le nombre de voisins en communs entre *c* et *n*. Ce paramètre représente la proximité entre les deux nœuds *c* et *n*.
– ***Fils(c)*** : le nombre de fils que le nœud $c$ a déjà associés. Lorsque deux nœuds *c1* et *c2* ont le même nombre de voisins en commun avec le nœud *n* c'est à dire $Commun(c1,n) = Commun(c2,n)$, *n* choisit de

s'associer avec le nœud *c* ayant le moins de fils. Ce critère permet d'assurer une meilleure linéarité du réseau en minimisant les formations en étoile et de se rapprocher autant que possible de la topologie physique étudiée.

- $-$ $\sum$ ***Voisin(n)*** : Nombre total de voisins de *c* et *n*. Lorsque deux nœuds *n1* et *n2* ont les deux premiers critères égaux avec le nœud $c$, celui avec le voisinage le plus réduit sera associé en premier. Ce nœud est en fait un nœud d'extrémité et a moins de chance de s'associer au prochain tour. Ceci permet ainsi de résoudre le problème d'apparition de nœuds orphelins rencontré dans ZigBee.
- $+$ ***MG(RSSI(c,n))*** : la moyenne glissante de la valeur du RSSI des paquets reçus par le nœud $n$ du nœud $c$. Lorsque les trois premiers métriques deux nœuds *n1* et *n2* sont égaux avec le nœud $c$, le nœud le plus proche (ayant la meilleure valeur de MG) sera associée en premier.

Ces critères sont condensés sous la forme d'une seule fonction (voir équation (2.1)) grâce à des pondérations de tel sorte à garder la priorité de chaque métrique et permettre de calculer le coût d'un lien $L(c, n)$.

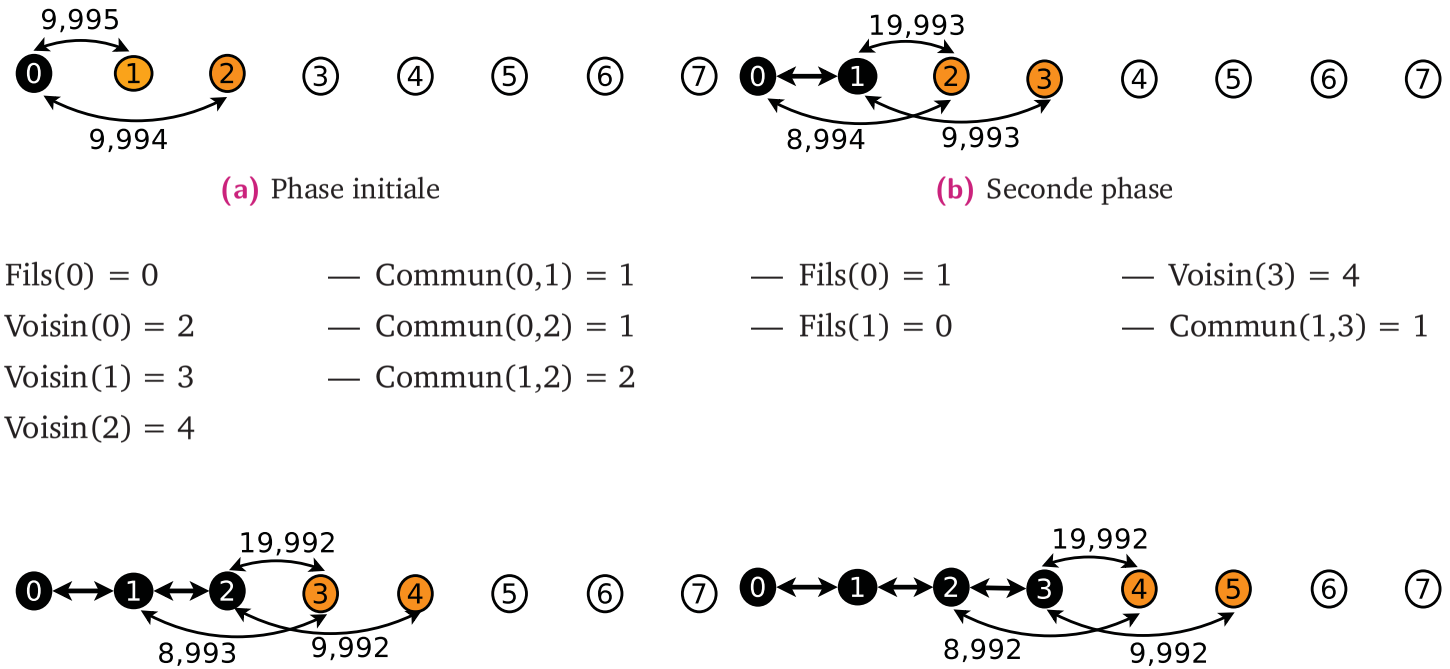
$$\begin{aligned} Objectif(c,n) = \quad & \alpha \times \text{Commun}(c,n) - \text{Fils}(c) \\ & -\beta \times (\text{Voisins}(c) + \text{Voisins}(n)) + MG(RSSI(c,n)) \\ & \text{avec } \alpha = 10,\ \beta = \tfrac{1}{1000},\ \gamma = \tfrac{1}{100000} \end{aligned} \tag{2.1}$$

Pour mieux évaluer le coût du lien, nous avons agrégé les trois critère en utilisant des pondérations $\alpha$, $\beta$ et $\gamma$. Les cœfficients $\alpha$, $\beta$ et $\gamma$ sont choisis de telle sorte qu'un critère est influent si et seulement si tous les critères d'ordre supérieur sont égaux pour les associations comparées et permettent ainsi d'agréger et de manipuler tous les critères dans une même estimation du coût. Pour cela, $\alpha$ doit être supérieur au plus grand nombre de voisins en commun possibles et $\beta$ doit être choisi tel que $\beta \times (\text{voisins}(c) + \text{voisins}(n)) < 1$.

Dans les schémas de la figure 2.11, nous montrons un exemple de construction d'une topologie linéaire avec notre algorithme *DiscoProto*.

À l'état initial de notre construction, le puits du réseau est choisi comme la racine de l'arbre et démarre la phase de découverte et de construction de la topologie.

Les différents liens entre le puits et chacun de ses nœuds voisins appelés *nœuds candidats* sont évalués. Dans la figure 2.11a la valeur $Objectif(0,1)$ du lien entre le puits et le nœud $1$ est égale à $9,995$ et celle avec le nœud $2$ est égale à $9,994$. Ainsi, le puits choisit le nœud $1$ qui deviendra ainsi un nœud de l'arbre (ou nœud connecté) et les différents liens avec son voisinage ne faisant pas partie de l'arbre seront évalués au prochain tour pour déterminer quel lien aura la meilleur valeur $Objectif$.




**(a)** Phase initiale

- Fils(0) = 0
- Voisin(0) = 2
- Voisin(1) = 3
- Voisin(2) = 4
- Commun(0,1) = 1
- Commun(0,2) = 1
- Commun(1,2) = 2

**(b)** Seconde phase

- Fils(0) = 1
- Fils(1) = 0
- Voisin(3) = 4
- Commun(1,3) = 1

**(c)** Troisième phase

- Voisin(1) = {2,3}
- Voisin(2) = {1,3,4}
- Voisin(3) = {1,2,4,5}

**(d)** Quatrième phase

- Commun(1,2) = 1
- Commun(1,3) = 1

**Fig. 2.11.:** Exemple de construction de la topologie avec l'approche centralisée de *DiscoProto* sur un segment linéaire

Cependant, notre première proposition n'est pas aisément implémentable en environnement de déploiement car elle nécessite de connaître l'ensemble des informations du réseau en un point central qui utilise l'algorithme *Disco-Proto*. Pour cette raison, nous avons étendu notre algorithme à une version distribuée.

## 2.4 L'approche distribuée de DiscoProto

Dans l'approche centralisée de notre algorithme, le lien avec la plus grande valeur de la fonction *Objectif* entre un nœud $c$ du sous-ensemble **C** et un nœud $n$ du sous-ensemble **N** de son voisinage est choisi. Cependant, puisque les nœuds de l'ensemble **C** sont déjà associés au réseau, ils pourront s'échanger des messages afin d'élire, durant un tour, le nœud $c$ ayant la meilleure valeur $objectif(c, n)$ dans un même voisinage. Cette compétition entre les nœuds $c$ du même sous ensemble **C** d'un voisinage est appelée **Challenge**.

Pour la mise en œuvre de la version distribuée de **DiscoProto**, nous avons considéré que le PAN coordinateur (ou puits) est le seul nœud du réseau qui est initialement connecté. Cela permet de construire une topologie qui démarre toujours à partir du PAN coordinateur. Le déroulement du protocole **DiscoProto** passe par quatre phases que sont :

— La découverte de voisinage,

— La collecte des fils,

— La collecte de la taille des sous-arbres,

— La propagation des adresses.

Durant ces phases, les nœuds passent par différents états qui sont représentés sur la figure 2.12 à la page 87que nous détaillons ci-après. Pour le bon fonctionnement de notre protocole, nous définissons différents types de messages qui sont décrits dans le tableau 2.1. Ces messages sont échangés par les nœuds durant les différents états de transitions définis dans chaque phase de *DiscoProto*.

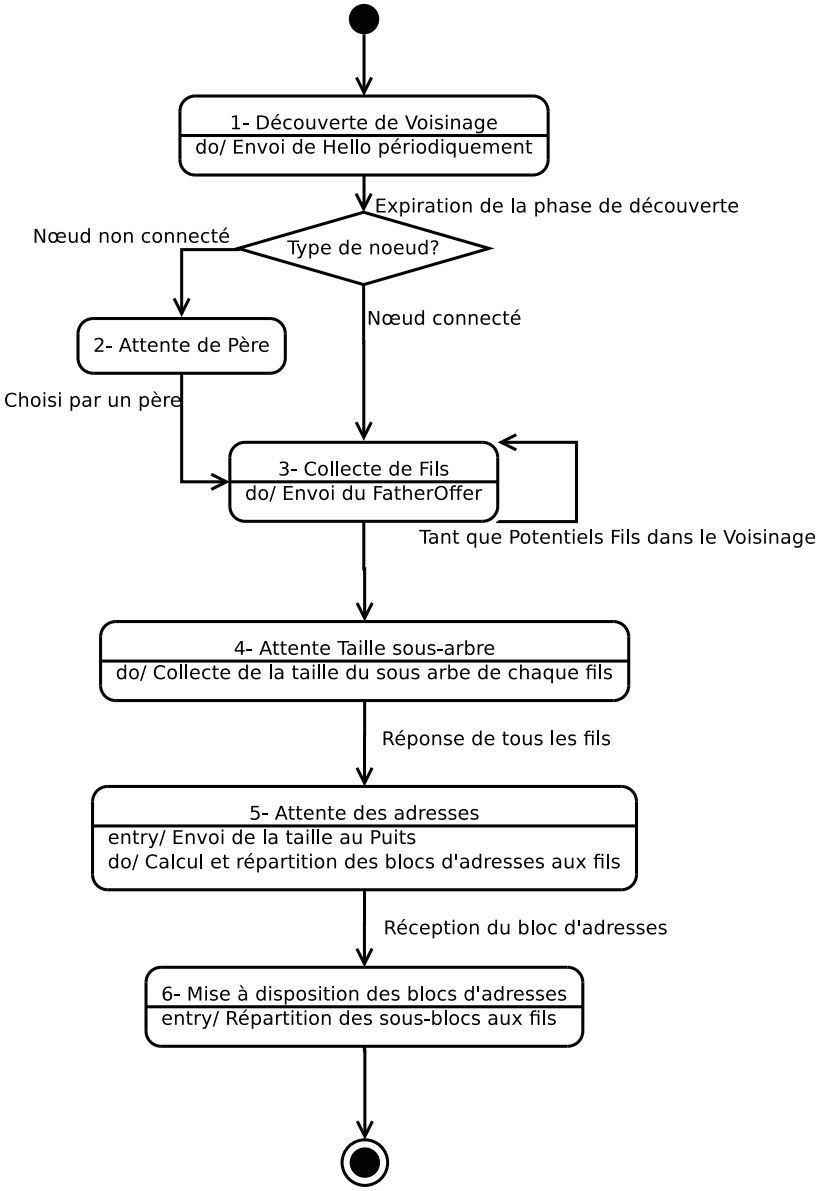



**Fig. 2.12.:** Différents états des nœuds avec *DiscoProto*

| Type de messages | Description | Phase |
|---|---|---|
| HELLO | Diffusé dans le voisinage pour annoncer sa présence | Découverte de voisinage |
| FatherOffer | Diffusé par un nœud connecté vers son voisinage non connecté et contenant son nombre de fils et la somme de ses voisins | Collecte des nœuds fils |
| SonOffer | Réponse à un *FatherOffer* reçu d'un nœud non connecté et contenant la valeur de la fonction $Objectif$ calculée par un nœud non connecté | Collecte des nœuds fils |
| ChallengeOffer | Diffusé par un nœud connecté vers son voisinage connecté et contenant l'Objectif choisi. Permet de choisir le nœud connecté ayant la meilleur valeur de l'Objectif (offre d'association) dans un même voisinage | Collecte des nœuds fils |
| ChallengeOfferResponse | Réponse d'un nœud connecté après la réception d'un *ChallengeOffer* contenant une valeur $Objectif$ plus petite | Collecte des nœuds fils |
| AssociationAccept | | Collecte des nœuds fils |
| AssociationAck | | Collecte des nœuds fils |
| PropaSons | | Propagation de la taille des sous-arbres |
| PropaAddr | | Propagation des plages de blocs d'adresses |

**Tab. 2.1.:** Différents types de messages de *DiscoProto*

| id_Son | addr_Node | Relation | RSSI | AGE |
|---|---|---|---|---|
| | | | | |

**Tab. 2.2.:** Format de la table de voisinage

### 2.4.1 Phase de découverte de voisinage

Au démarrage des nœuds du réseau, le protocole *DiscoProto* commence par la phase de ***Découverte de voisinage*** durant laquelle les nœuds sont dans un état appelé aussi *Découverte de voisinage*. Durant cette phase, les nœuds diffusent et collectent des messages *HELLO* dans leur voisinage afin de mettre à jour leur table de voisinage (Voir table 2.2). La table de voisinage contient ,pour chaque entrée, l'identifiant du nœud voisin (id_Son), son adresse logique (addr_Node), sa relation avec le nœud (Relation), la moyenne glissante de son RSSI (RSSI) (Received Signal Strength Indicator), l'age de l'entrée (AGE) représentée par la date de création ou de mise à jour de l'entrée de la table. Nous avons défini, pour le protocole *DiscoProto*, quatre types de relation : PERE, FILS, GRANDPERE, AUTREVOISIN. Le champs *AGE* permet durant toute la durée de vie de garder dans la table de voisinage que les nœuds voisins disponibles et de supprimer les entrées des nœuds voisins ayant atteint l'age maximum (cinq secondes par défaut). Lors de cette phase de découverte de voisinage, seuls les champs id_Son, Relation, RSSI,AGE sont renseignés avec comme valeur par défaut des champs relation égale à $AUTREVOISIN$. Les valeurs des champs addr_Node, relation seront mises à jour lors des phases de collecte de fils et de distribution des adresses.

Cette étape est très importante car le calcul de la valeur de *Objectif* dépend fortement de la connaissance du voisinage. À la fin de cette phase (expiration du temporisateur de découverte de voisinage), on considère que chaque nœud du réseau a connaissance de son voisinage.

### 2.4.2 Phase de collecte de fils

Après la phase de découverte de voisinage, *DiscoProto* entame la phase de ***Collecte des fils***. Durant cette phase, les nœuds connectés $c$ de l'ensemble **C** passent à l'état de *Collecte des fils* où ils diffusent des messages appelés *FatherOffer* contenant le nombre de nœuds voisins et le nombre de nœuds fils. Le but de cet état est que chaque nœud connecté ait connaissance de tous les fils qu'il pourrait associer. Au démarrage du réseau, le puits est le seul nœud connecté. Les nœuds non connectés $n \in$ **N** passent à l'état d'*Attente de père* où ils acceptent et traitent tous les messages *FatherOffer* venant des nœuds connectés de leur voisinage. L'algorithme 2.2 à la page 92 décrit les

traitements effectués par un nœud connecté durant la phase de collecte de fils.

Lorsqu'un nœud non connecté $n$ reçoit un message *FatherOffer* d'un nœud connecté $c$, il calcule la valeur $Objectif(c,n)$ du lien $L(c,n)$ et répond par un message *SonOffer* contenant la valeur $Objectif$. L'algorithme 2.3 à la page 93 décrit les traitements effectués par un nœud non connecté après la réception d'un message *FatherOffer*.

Chaque nœud connecté collecte tous les messages *SonOffer* de son voisinage jusqu'à l'expiration d'un temporisateur *WAIT_FOR_SON_OFFER*. À la fin du temporisateur, le nœud connecté choisit la plus grande valeur $Objectif$ reçue (son meilleur nœud fils possible) et diffuse un message *ChallengeOffer* contenant la valeur $Objectif$ dans le voisinage. Il passe ensuite à l'état *Attente d'un Challenge*. Lors de la diffusion du *ChallengeOffer*, seul les nœuds connectés qui sont à l'état de *Attente d'un Challenge* acceptent et traitent un *ChallengeOffer* venant d'autres nœuds jusqu'à l'expiration d'un temporisateur *CHALLENGE_TIMEOUT*. Durant l'étape du Challenge, le message *ChallengeOffer* est relayé localement jusqu'à un maximum de trois sauts. Cette approche de propagation du *ChallengeOffer* permet de choisir un maximum local contrairement à l'approche centralisée de *DiscoProto*.

Lorsqu'un nœud connecté $c2 \in C$ avec une valeur $Objectif2$ reçoit un *ChalengeOffer* contenant une valeur $Objectif1$ d'un autre nœud connecté $c1 \in C$. Si $Objectif1 < Objectif2$ alors $c2$ a une meilleure offre que $c1$ et envoie un message *ChalengeOfferResponse* pour annuler la phase de collecte de fils de $c1$. Dans le cas où $c1$ a une meilleure offre, alors $c2$ annule sa propre phase de collecte de fils et en recommence une nouvelle à la fin du temporisateur *CHALLENGE_TIMEOUT*. À l'expiration du temporisateur *CHALLENGE_TIMEOUT*, un seul nœud connecté est gagnant du challenge et passe à l'état d'*Association* où il envoie un message *AssociationAccept* au nœud non connecté qui a été choisi comme meilleur fils possible. Ce dernier répond par un message *AssociationAck* et devient alors un nœud connecté avant de démarrer à son tour une phase de collecte de fils. L'algorithme 2.5 à la page 94 décrit le traitement effectué par un nœud connecté $c$ à la réception d'un message *ChallengeOffer*. Tous les nœuds connectés recommencent une nouvelle phase de collecte de fils à la fin de l'association. Un nœud connecté termine sa phase de collecte de fils lorsqu'il n'existe plus de nœuds non connecté

dans son voisinage. Durant la phase de collecte des fils, les nœuds connectés qui sont suffisamment éloignés de telle sorte qu'ils ne peuvent pas recevoir leur message *ChalengeOffer* respectif peuvent faire des associations de façon indépendantes durant les mêmes cycles (voir plus loin à la page 96) et ainsi effectuer la collecte de fils parallèlement.

La figure 2.13 décrit le diagramme de séquence de la phase de collecte avec les différents messages *HELLO*, *FatherOffer*, *SonOffer*, *ChalengeOffer*, *AssociationAccept* et *AssociationAck* échangés par les nœuds connectés et non connectés d'un même voisinage.

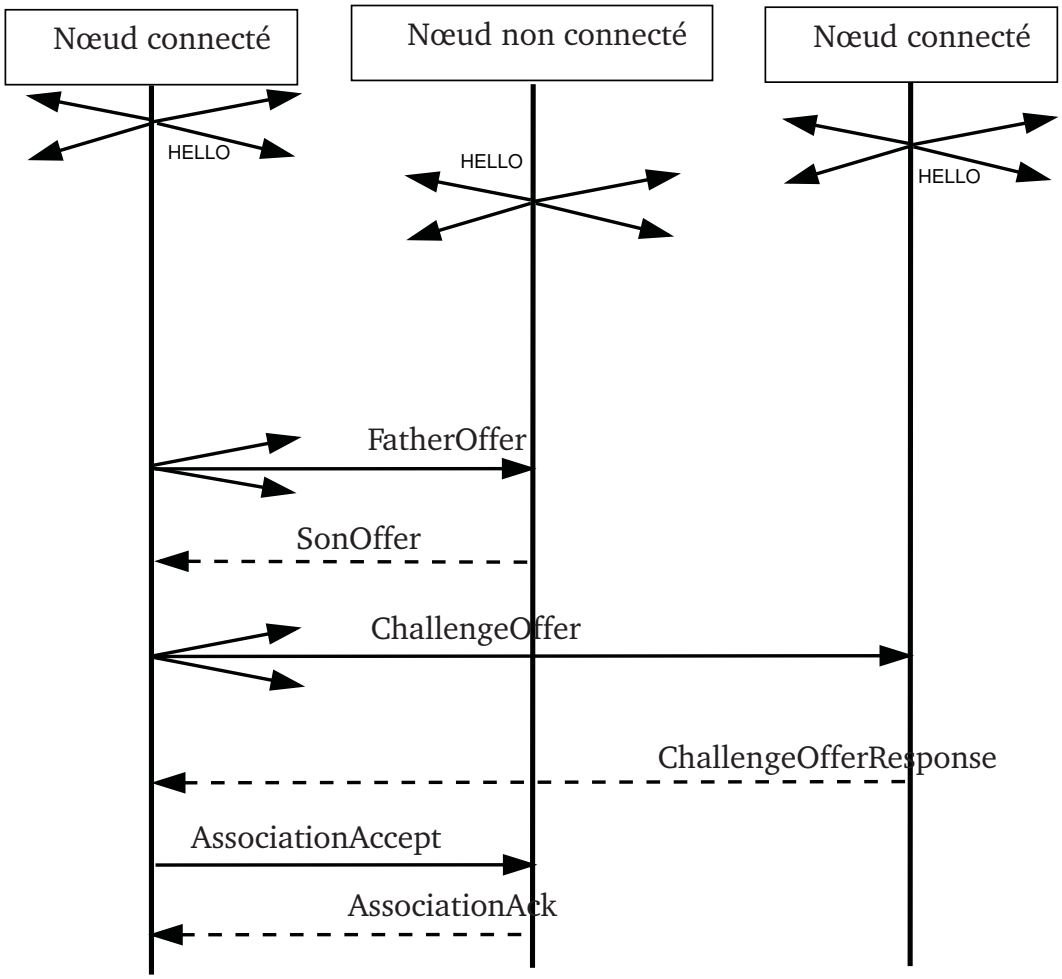



**Fig. 2.13.:** Diagramme de séquence de la phase de collecte de fils

**Algorithme 2.2:** Collecte de Fils

**Result:** Collecter de nouveaux fils, terminer si pas de candidats trouvés.
**Input :** Un ensemble de voisins : $neighbors$
**Input :** $MACAddr$ l'adresse MAC du nœud
**Input :** $COLLECTING\_SONS\_TIMEOUT$ le délai d'attente des réponses *SonOffer* des fils à un *FatherOffer* envoyé
**Input :** $CHALLENGE\_TIMEOUT$ le délai pour recevoir une réponse à un *ChallengeOffer*

```
1  state ← WAIT_FOR_SON_OFFER ;
2  k ← 3 ;        // Nombre de sauts maximum de diffusion du Challenge
3  while state = WAIT_FOR_SON_OFFER do
4      offer ← ∅ ;          // mémoriser et stocker la meilleure valeur
        Objectif reçue
5      fatherOffer ← (macAddr,neighbors,sons);
6      FatherOffer (macAddr,fatherOffer);
7      Broadcast(FatherOffer);
8      Sleep(COLLECTING_SONS_TIMEOUT) ;          // Réception des
        SonOffer venant des potentiels fils du voisinage
9      if offer ≠ ∅ then
           // Au moins un fils a répondu par un SonOffer
10         better ← false ;
11         ChallengeOffer (macAddr,offer,k);
12         Broadcast (ChallengeOffer) ; // Demander aux nœuds connectés
            voisins s'ils ont une meilleure offre
13         state ← WAIT_FOR_CHALLENGE ;
14         Sleep(CHALLENGE_TIMEOUT) // Collecter les réponses au
               ChallengerOffer
15         if better = false then
16             state ← ASSOCIATION ;
17             AssociationAccept ((macAddr,offer.MACaddr));
18             Unicast (AssociationAccept);
19         state ← WAIT_FOR_SON_OFFER
20     else
21         state ← WAIT_FOR_SON_SIZE
```

| Variables | Description |
|---|---|
| macAddrr | Adresse mac d'un nœud |
| neighbors | Tableau contenant la liste des nœuds voisins |
| sons | Nombre total de fils d'un nœud connecté (potentiel père) |
| fatherOffer | Structure de données contenant la liste des voisins, le nombre de fils et l'adresse mac du nœud potentiel père |
| offer | Structure de données contenant la meilleure valeur objectif, et l'adresse mac du nœud potentiel fils |
| sonOffer | Structure de données contenant la valeur $objectif$ et l'adresse mac du nœud potentiel fils émetteur |

| Fonctions | Description |
|---|---|
| FatherOffer | Message de diffusion contenant le fatherOffer du nœud potentiel père |
| SonOffer | Message unicast contenant le sonOffer du nœud potentiel fils |
| ChallengerOffer | Message contenant la valeur $objectif$ choisie du nœud potentiel père et le nombre maximum de sauts pour le relayage du message |
| AssociationAccept | Message contenant le fatherOffer du nœud potentiel père |
| Broadcast | Méthode pour diffuser un message donné |
| Unicast | Méthode pour envoyer un message vers un nœud |

**Tab. 2.3.:** Variables et fonctions utilisées dans les algorithmes *DiscoProto*

**Algorithme 2.3:** Traitement à la réception d'un FatherOffer (action à l'état *Attente de Père*)

**Result:** Réaction suite à la réception d'un FatherOffer.
**Input :** FatherOffer venant d'un nœud connecté du voisinage
**Output :** SonOffer à destination de l'émetteur du FatherOffer

```
1 if state = WAIT_FOR_FATHER_OFFER then
2 |   objectif ← Objectif
  |    (fatherOffer.sons,fatherOffer.neighbors,sons,neighbors);
3 |   offer ← (macAddr,objectif);
4 |   SonOffer (macAddr,offer);
5 |   Unicast (SonOffer);
```

**Algorithme 2.4:** Traitement à la réception d'un SonOffer

**Result:** Réaction suite à la réception d'un SonOffer.
**Input :** SonOffer venant d'un nœud non connecté du voisinage

```
1  repeat
2      if SonOffer = ∅ then
3          return ;
4      sonOffer ← SonOffer.offer;
5      if offer = ∅ then
           // réception de la première offre
6          offer ← sonOffer ;
7          return ;
8      if state = WAIT_FOR_SON_OFFER ∨ sonOffer.objectif < offer.objectif
        then
           // Meilleur offre reçu
9          offer ← sonOffer ;
10 until Expire (COLLECTING_SONS_TIMEOUT);
```

**Algorithme 2.5:** Réception d'un ChallengeOffer

**Result:** Traitement à la réception d'un message ChallengeOffer
**Input :** $MACAddr$ l'adresse MAC du nœud
**Input :** $CHALLENGE\_TIMEOUT$ le délai pour recevoir une réponse à un *ChallengeOffer*

```
1  if state = WAIT_FOR_CHALLENGE then
2      challengeOffer ← ChallengeOffer.offer;
3      if challengeOffer.objectif > offer.objectif then
4          better ← true ;
5      if challengeOffer.objectif < offer.objectif then
6          NextHop ← ChallengeOffer.path.last;
7          Dest ← challengeOffer.MACAddr;
8          ChallengeOfferResponse ((macAddr,offer, Dest));
9          Unicast (ChallengeOfferResponse,NextHop);
10     if ChallengeOffer.hop ≥ 1 offer.objectif ∨ macAddr ∉
        ChallengeOffer.path then
11         ChallengeOffer.hop ← ChallengeOffer.hop - 1;
12         ChallengeOffer.path ← ChallengeOffer.path ∪ macAddr ;
13         Broadcast (ChallengeOffer);
```

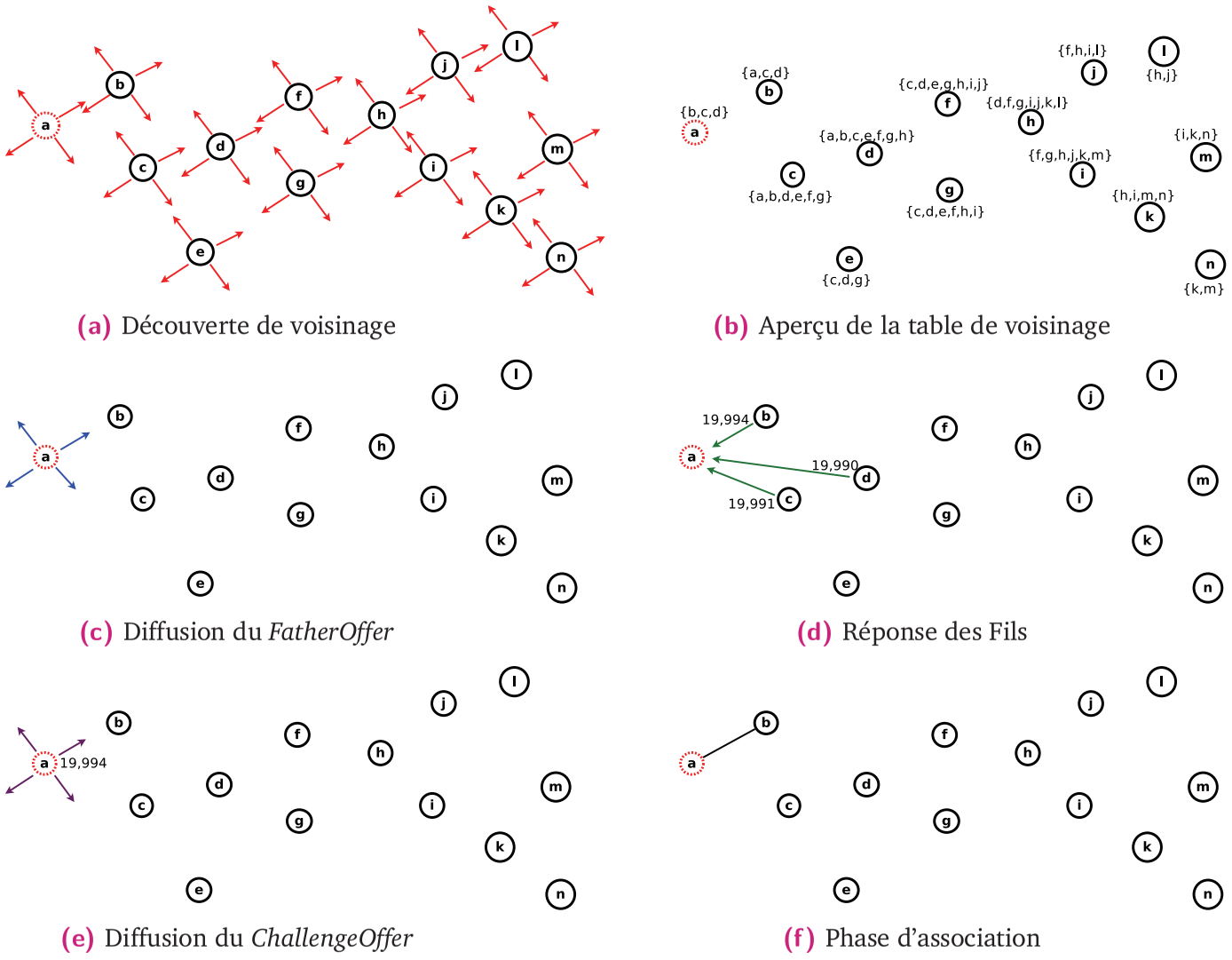
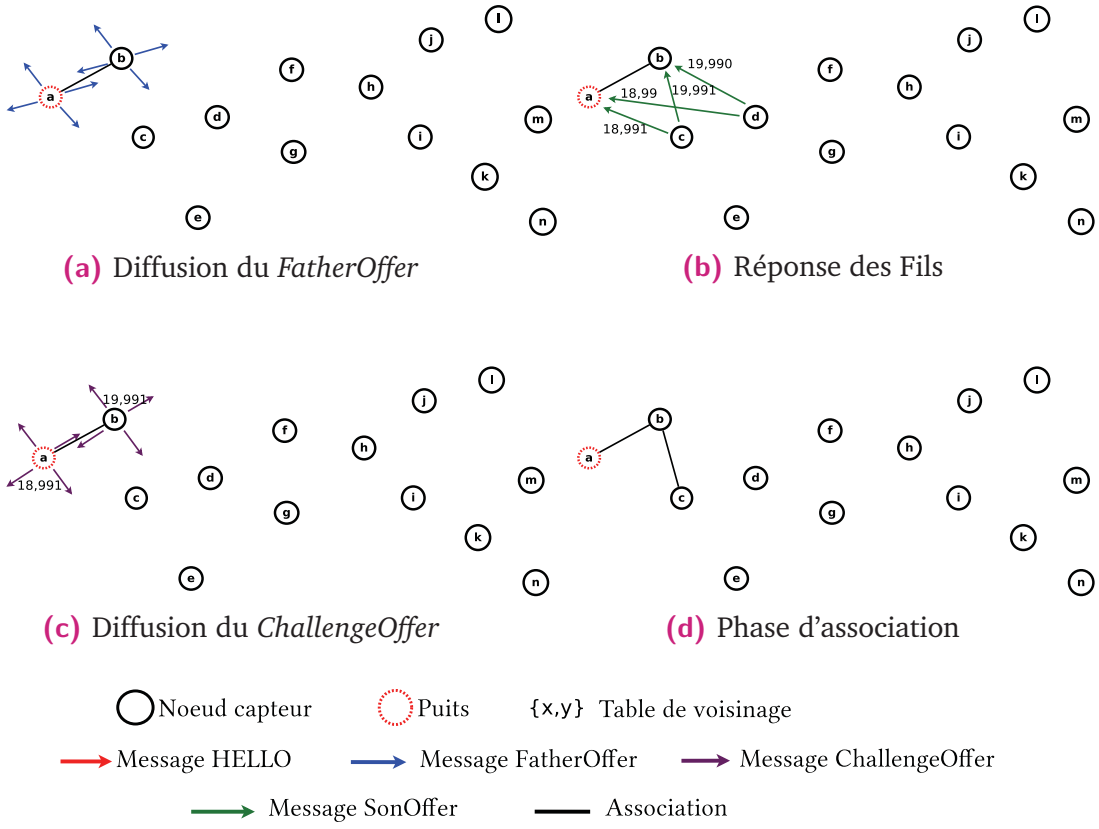



**Fig. 2.14.:** Étapes de DiscoProto au démarrage du réseau

La figure 2.14 à la page 95 est un exemple simplifié montrant les différentes étapes de la formation d'un réseau avec *DiscoProto*. À la fin de la phase de découverte de voisinage (figure 2.14a), le puits, représenté par le nœud $a$, commence la phase de collecte de fils (figure 2.14c) en diffusant un message *FatherOffer*. Seuls les nœuds de son voisinage à savoir $b, c$ et $d$ répondent par un message *SonOffer*. Le nœud $b$, ayant envoyé la meilleure valeur $Objectif$ (figure 2.14d), sera choisi pour l'étape d'association à la fin du challenge (figure 2.14f).

Après l'association du nœud $b$, ce dernier fait partie de l'ensemble connecté et démarre avec le PAN coordinateur une nouvelle phase de collecte de nœuds fils (voir figure 2.15). Durant cette phase, le nœud $b$ gagne le challenge avec le puits et poursuit l'association avec le nœud $c$.



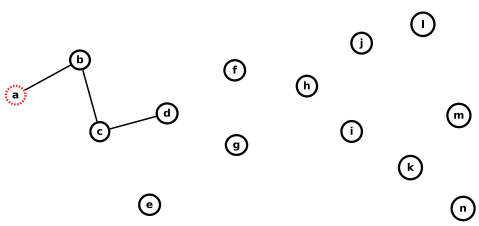

**(a)** Diffusion du *FatherOffer* **(b)** Réponse des Fils

**(c)** Diffusion du *ChallengeOffer* **(d)** Phase d'association

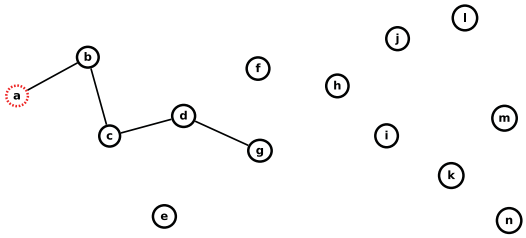
**Fig. 2.15.:** Étapes de DiscoProto au deuxième cycle

La figure 2.16 à la page 97 montre les cycles suivants de la phase de collecte de fils. Nous remarquons dans ces différents schémas que le nœud $e$ n'est toujours pas associé. En effet, le nœud connecté $c$ avec la valeur de l'$Objectif$ reçue du nœud $e$, perd toujours le challenge face aux autres nœuds. Cepen-

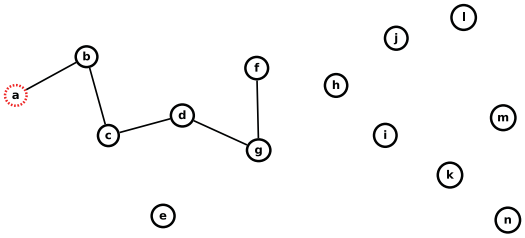
dant, le nœud $c$ finira par associer ce nœud $e$ lorsque son challenge ne sera pas en compétition avec un autre nœud de son voisinage comme l'illustre la figure 2.17



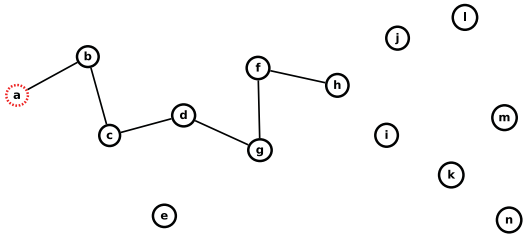

**(a)** Troisième cycle de collecte de fils




**(b)** Quatrième cycle de collecte de fils




**(c)** Cinquième cycle de collecte de fils



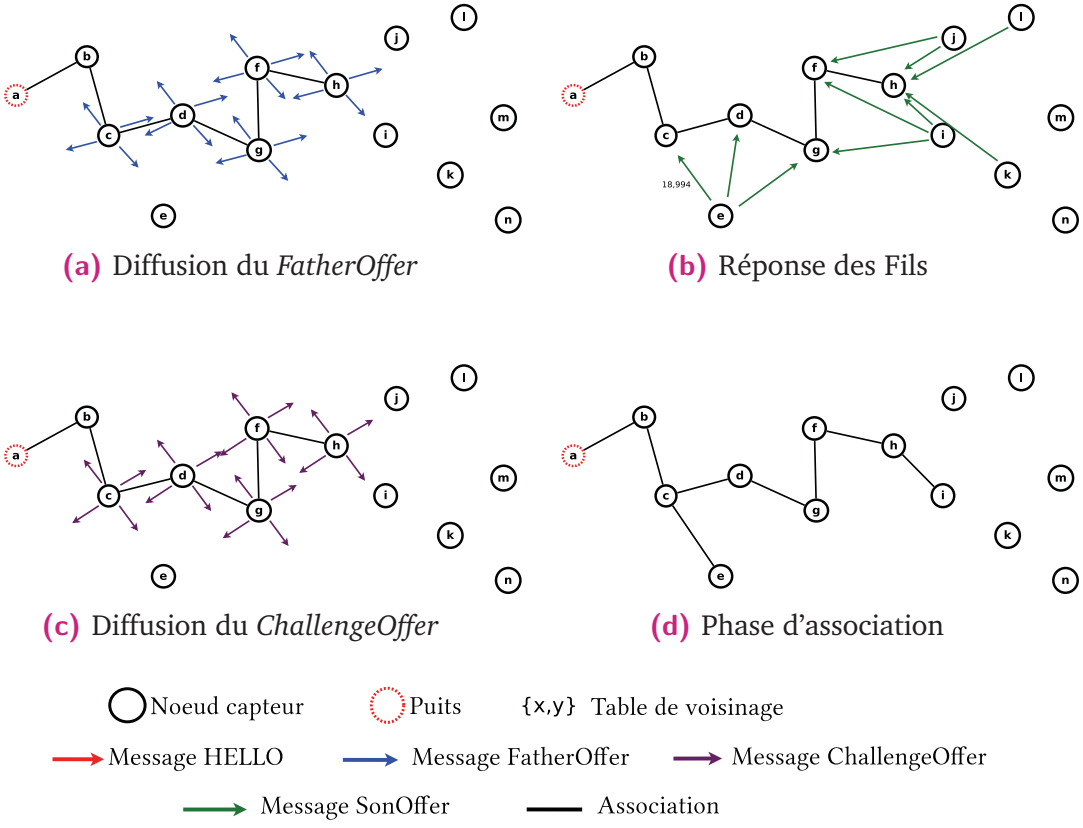

**(d)** Sixième cycle de collecte de fils

**Fig. 2.16.:** Cycles de Collecte de nœuds Fils

La figure 2.18 à la page 98 montre un exemple de topologie formée après la phase de collecte de fils. Les numéros sur les liens représentent les ordres d'association des nœuds durant la phase de collecte de fils.

### 2.4.3 Phase de propagation de la taille des sous arbres

Un nœud qui n'a pas de nœuds non connectés à portée démarre la phase de ***Propagation d'adresses*** et passe à l'état d'*Attente de la taille des fils*. Durant cet état, chaque nœud connecté attend de recevoir, de l'ensemble de ses fils, la taille de leur sous arbre (contenue dans un message *PropaSons*). Les nœuds connectés qui n'ont pas de nœuds fils sont à une extrémité. Á l'état *Attente de la taille des fils*, ils envoient donc à leur nœud père un message *PropaSons* contenant la valeur 1. Après avoir envoyé la taille de leur sous arbres, chaque nœud connecté passe à l'état *Attente d'adresses*. Avec cette approche de propagation de la taille des sous arbres initiée depuis les nœuds




**(a)** Diffusion du *FatherOffer* **(b)** Réponse des Fils

**(c)** Diffusion du *ChallengeOffer* **(d)** Phase d'association

**Fig. 2.17.:** Étapes de DiscoProto au septième cycle

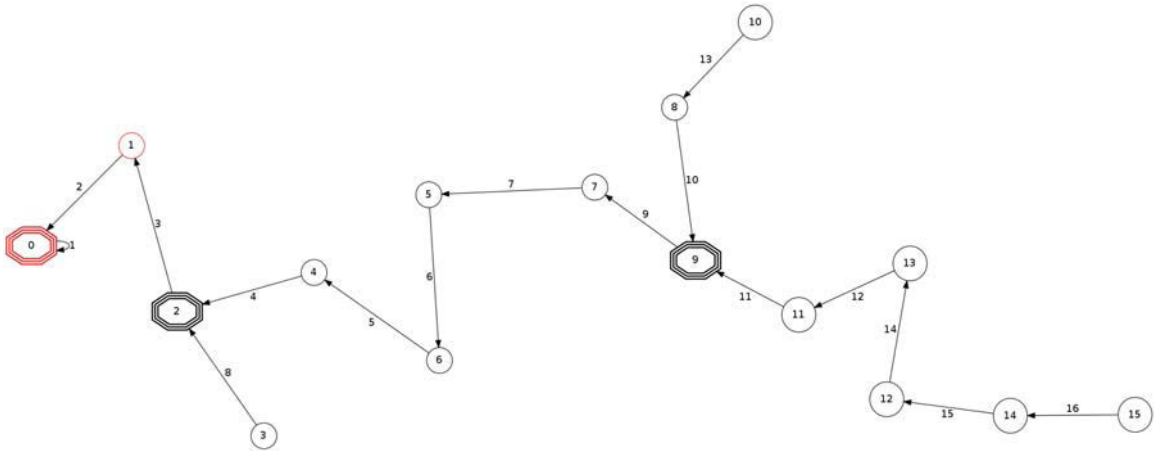



**Fig. 2.18.:** Exemple de topologie générée par *DiscoProto*

d'extrémité. Lorsque le puits reçoit les messages *PropaSons* de ses nœuds fils, tout nœud du réseau connait la taille du sous-arbre de chacun de ses fils.

Durant les phases de découverte de voisinage et de collecte des nœuds, des messages *HELLO*, *FatherOffer* et *ChallengeOffer* sont diffusés périodiquement dans le réseau. Ceci peut occasionner des collisions dans un même voisinage lorsque des nœuds ont des cycles identiques d'émission. Les collisions durant la phase de découverte de voisinage engendre des tables de voisinages ne contenant pas tous les nœuds voisins. Les collisions des *FatherOffer* durant un cycle de collecte de fils peuvent occasionner la nom réception de ces messages par les nœuds potentiels fils. Ainsi, ces nœuds ne pourront pas calculer et envoyer leur offre à ces nœuds émetteurs de *FatherOffer*. Les collisions des *ChallengeOffer* peuvent causer la non réception des challenges dans un même voisinage. De ce fait, à la fin des temporisateurs du challenge, les nœuds émetteurs des *ChallengeOffer* vont penser chacun qu'ils ont la meilleure valeur *Objectif* et effectuer leur association.

Pour atténuer ces risques de collisions, nous avons utilisé des temporisateurs tirés aléatoirement afin de désynchroniser les cycles d'émission des messages *HELLO*, *FatherOffer* et *ChallengeOffer*.

### 2.4.4 Phase de propagation des adresses

À la réception des sous arbres de tous ses nœuds fils, le PAN coordinateur calcule l'espace d'adressage optimal $TailleBlocAdresse$ en fonction de la taille du réseau grâce à l'équation (2.2) où $Fskip$ représente le nombre de nouveaux nœuds que chaque nœud du réseau pourra ajouter ultérieurement. Cependant pour les besoins de calcul et de la répartition des adresses, nous avons fixé par défaut la valeur $Fskip$ à deux. En effet, lorsque la connectivité est perdue à cause de la disparition de nœuds dans un RdCSFL, chaque nœud concerné aura besoin d'une adresse de réserve pour rétablir la disponibilité du réseau. Le choix d'une valeur de $Fskip$ très grand permettra d'ajouter certes une longue branche de nœuds tout en évitant de solliciter le PAN coordinateur. Cependant, ce choix occasionnera l'épuisement très rapide de l'espace d'adressage et augmentera la possibilité d'avoir des nœuds orphelins dans le réseau lorsque l'ajout de nouveaux se fait sur une même partie du réseau.

Ensuite, le PAN coordinateur initie la phase de propagation des adresses et envoie à chacun de ses fils un message *PropaAddr* contenant le bloc d'adresses calculé proportionnellement à la taille leur sous-arbre.

$$TailleBlocAdresse = \sum_{i=1}^{n} TailleSousArbreFils(i) \times (Fskip + 1) \tag{2.2}$$

Chaque nœud connecté qui reçoit un message *PropaAddr* contenant le bloc $[a, b]$ procède comme suit :

1. il choisit l'adresse $a$ du bloc comme sa propre adresse,
2. ensuite, il garde le bloc $[a + 1, a + Fskip]$ pour l'ajout de futurs nœuds
3. enfin, il partage le bloc restant $[a + Fskip + 1, b]$ à ses fils proportionnellement à la taille leur sous-arbre en utilisant des messages *PropaAddr*.

La figure 2.19 décrit le diagramme de séquence lors de la collecte des tailles des sous-arbre et de la distribution des adresses grâce aux différents messages *PropaAddr*, *PropaSize*.

À la fin de cette phase d'adressage, chaque nœud du réseau a une adresse logique ainsi que $Fskip$ adresses en réserve et connaît aussi les adresses de son nœud parent et de chacun de ses fils.

Dans la Figure 2.20a, les nœuds remontent progressivement la taille de leur sous-arbre en commençant par les nœuds périphériques jusqu'au nœud puits (nœud $a$). Le puits reçoit la valeur $15$ comme taille du sous-arbre de son nœud fils $b$ et calcule le bloc d'adresses (espace d'adresses) nécessaire pour l'intégralité du réseau soit $TailleBlocAdresse = 15 \times 3 = 45$ (Voir équation 2.2 à la page 100). Pour cet exemple la valeur de la réserve d'adresses est égale à deux ($Fskip = 2$). Le puits démarre alors la distribution des adresses proportionnellement à la taille des sous-arbres (figure 2.20b). Cette distribution se poursuit progressivement jusqu'aux nœuds de périphérie.

Lorsque qu'un nœud du réseau reçoit un bloc d'adresse $[a, b]$, il procède comme suit.

1. Il choisit la première adresse du bloc comme sa propre adresse logique qu'il utilisera pour le routage des données vers le puits ;

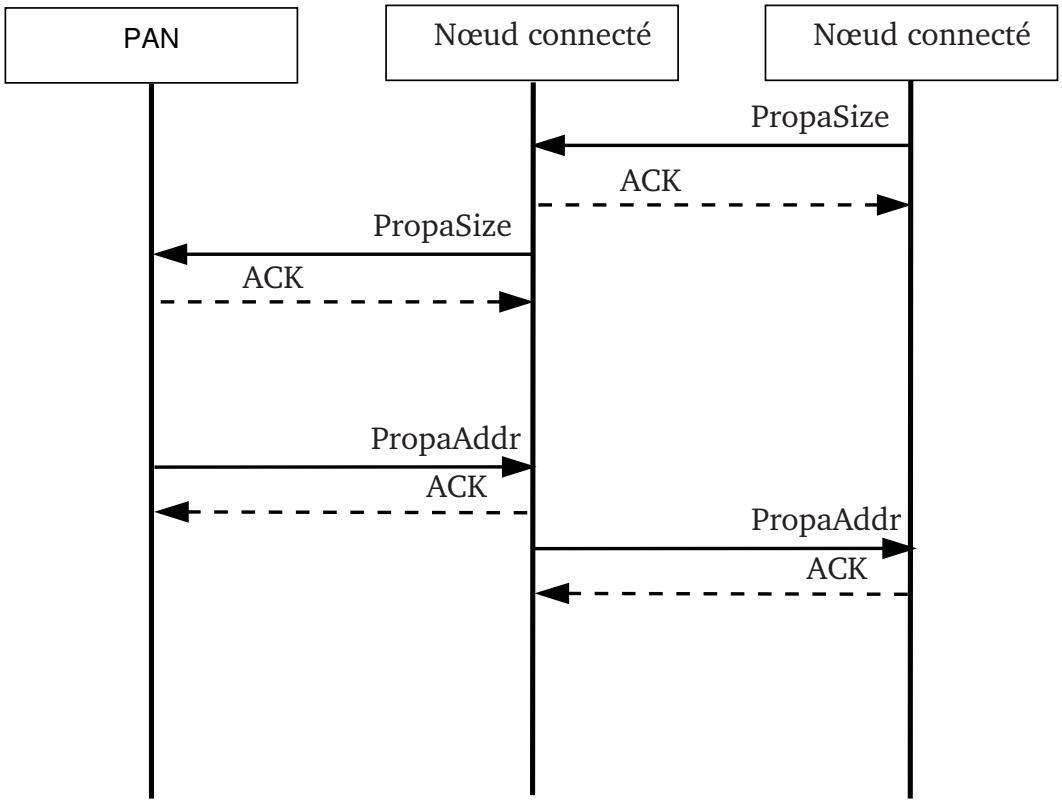


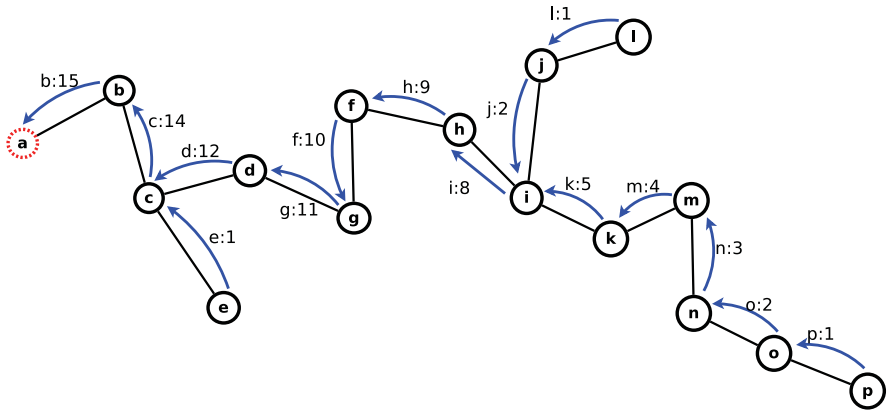

**Fig. 2.19.:** Diagramme de séquence de la phase de collecte des tailles des sous-arbres et de propagation des adresses

2. Il réserve le bloc $[a+1, a+Fskip]$ où $Fskip$ est la réserve locale pour l'ajout de nouveaux nœuds ;
3. Il distribue le bloc restant $[a+Fskip+1, b]$ à ses nœuds fils proportionnellement à la taille de leur sous-arbre.

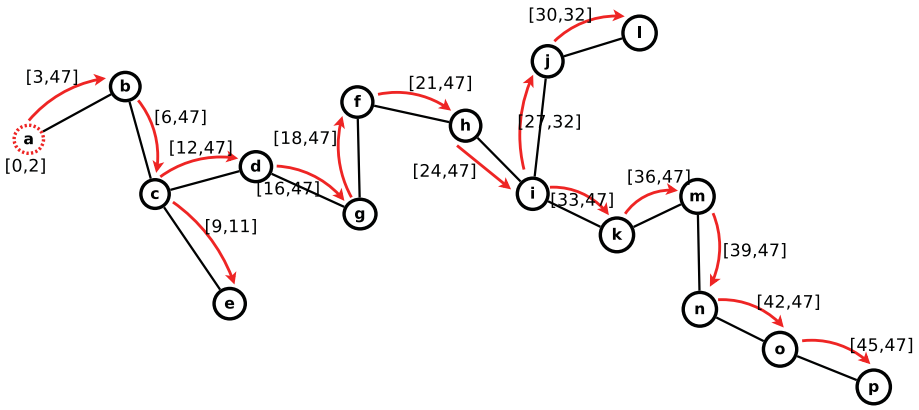
## 2.5 Routage de données dans DiscoProto

Avec son approche d'adressage distribuée des nœuds, *DiscoProto* fournit un mécanisme de routage sans utilisation de protocole de routage dynamique supplémentaire. Lors de la phase de distribution des adresses, chaque nœud mémorise une table d'adresses contenant les début et fin des blocs d'adresses attribués à chacun de ses fils.

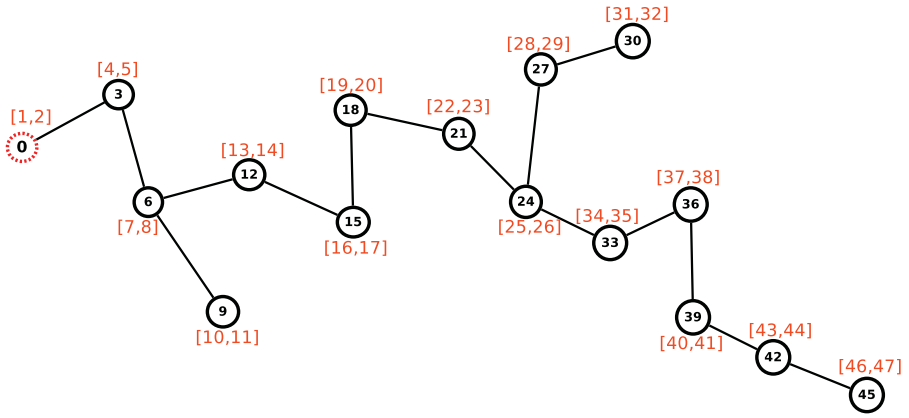
Le tableau 2.4 décrit le format de la table d'adresses de chaque de nœud du réseau.




**(a)** Collecte de la taille des sous-arbres




**(b)** Distribution des adresses




**(c)** Réseau avec adresses logiques et réserves d'adresses

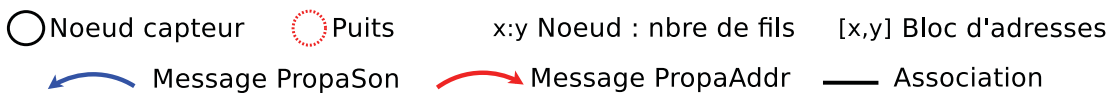



**Fig. 2.20.:** Étapes d'adressage des nœuds

| id_Son | addr_Node | start_Block | end_Block |
|---|---|---|---|
| Adresse physique | Adresse logique | Adresse logique | Adresse logique |

**Tab. 2.4.:** Format de la table d'adresses

| id_Son | addr_Node | start_Block | end_Block |
|---|---|---|---|
| 4 | 9 | 9 | 11 |
| 5 | 12 | 12 | 26 |

**Tab. 2.5.:** Exemple de la table d'adresses du nœud 6 de la figure 2.20c

Lorsque qu'un nœud $x$ du réseau reçoit un paquet de données, le mécanisme de routage implémenté exécute les procédures suivantes :

1. Si le nœud $x$ est la destination, le paquet est accepté et transmis à la couche supérieure
2. Sinon, le nœud vérifie si la destination correspond à l'une des entrées de la table d'adresses. Si une correspondance est trouvée, le paquet est transmis au nœud fils de cette entrée.
3. Pour tout autre cas, le paquet est transmis au nœud père en direction du nœud Puits.

Pour assurer une disponibilité accrue du réseau K-redondant avec $K \geq 2$ et le relayage des données lorsqu'un nœud relais disparaît du réseau, notre mécanisme de routage permet à nœud de relayer les données à son nœud grand père en direction du puits lorsque son père n'est plus disponible dans sa table de voisinage. Pour ce faire, lors de la phase d'association d'un nœud, chaque nœud père inclut, dans le message *AssociationAccept*, l'identifiant de son propre nœud père. À la réception du message, le nœud fils fait une mise à jour de l'entrée correspondant de la table de voisinage en mettant la valeur du champs relation à $GRANPERE$. Ce qui permet lorsque l'entrée du nœud père est supprimée de la table de voisinage (age maximum de l'entrée atteint), le nœud transmet les données au nœud dont l'entrée relation correspond à $GRANPERE$.

## 2.6 Le générateur de topologie linéaire

Le développement de cet outil a été motivé par l'absence d'un générateur de topologies linéaires pour les réseaux de capteurs sans-fil dans le simulateur

Omnet++/Castalia. En effet, l'environnement de simulation propose un gestionnaire de mobilité qui fournit différents modes de déploiement (aléatoire, en grille sur $2$ et $3$ dimensions, ou manuel à partir d'un fichier de coordonnées). Pour nos travaux, nous avons implémenté un module qui permet le placement des nœuds aléatoirement sur une ligne pouvant comporter des branchements.

Notre générateur de topologies linéaires, développé en C++ (voir une partie du code dans l'annexe A), fournit deux modes de génération :

— Génération de topologies de taille donnée avec des longueurs de branches de nœuds différentes.

— Génération de topologies avec un nombre de branches donné de longueurs égales.

Les paramètres ci-dessous permettent de créer ces différentes topologies :

— Nombre de segments de lignes contenant des nœuds actifs $onLines$,

— Nombre de nœuds actifs par ligne $lineLength$,

— Nombre de segments de lignes contenant des nœuds inactifs $offLines$. Ce paramètre permet d'ajouter des branches entières de nœuds éteints qui seront actifs après une durée déterminée $sleepduration$,

— Nombre de nœuds inactifs $offNodes$. Ce paramètre permet d'ajouter des nœuds éteints qui seront actifs après une durée déterminée $sleepDuration$,

— Distance $d$ entre les nœuds,

— Delta distance $\Delta d$ tirée aléatoirement entre $[-\Delta d, \Delta d]$ et ajoutée à la distance $d$ entre deux nœuds,

— vibration $\Delta vibration$ tirée aléatoirement entre $[-\Delta vibration, \Delta vibration]$ qui représente l'angle de déviation du vecteur à chaque placement de nœuds,

— Probabilité de créer un branchement dans le réseau linéaire $forkProbability$,

— Angle de déviation tiré aléatoirement durant la création d'un nouveau branchement $\Delta angle$,

Dans la mise en œuvre de nos simulations, nous avons généré des topologies basées sur le tableau ci-dessus.

| Paramètres | Valeurs |
|---|---|
| Surface de déploiement | $1000\ m \times 1000\ m$, $2000\ m \times 2000\ m$, $3000\ m \times 3000\ m$, $4000\ m \times 4000\ m$ |
| Taille de réseau | $50$, $100$, $150$, $200$, $250$, $300$, $350$, $400$, $450$, $500$ nœuds |
| Distance entre les nœuds $d$ | $20$ mètres |
| Delta distance $\pm\Delta d$ | $\pm 2$ mètres |
| Delta vibration $\Delta vibration$ | $\pm 10$ degrés |
| Probabilité de branchement | $10\%$ |
| Angle de déviation | entre $40$ et $140$ degrés et $-40$ et $-140$ degrés |

**Tab. 2.6.:** Paramètres de génération des topologies linéaires

En annexe B, nous présentons quelques exemples de topologies générées et utilisées dans nos différentes simulations.

Afin de vérifier la linéarité de nos topologies, nous avons écrit un script en Ruby appelé *linearityCheck.rb* (voir Annexe D) qui vérifie que la topologie réseau générée respecte un seuil de linéarité en se basant sur la distance entre les nœuds définie dans les paramètres du générateur de topologie. Ce script nous a permis de ne fournir au simulateur que les topologies qui respectent les conditions de linéarité explorées.

## 2.7 Évaluation du protocole *DiscoProto*

Pour l'évaluation de *DiscoProto*, nous avons fait des simulations sur $1000$ topologies différentes linéaires respectivement de $50$, $100$, $150$, $200$, $250$ et nous allons étudier :

— La proximité de la topologie logique générée avec la topologie physique déployée, en comparant le nombre de disjonctions de la topologie logique avec le nombre de branchements de la topologie physique. Un branchement est un nœud de la topologie physique à partir duquel une nouvelle branche est créée. Une disjonction représente un nœud de la topologie logique avec plusieurs fils. Ces disjonctions sont dans la plupart des cas détectés dans le voisinage d'un branchement,

— La durée moyenne de l'association des nœuds du réseau et l'influence du nombre de branchements,

— L'efficacité du protocole à former une topologie de RdCSF linéaire avec le taux d'association qui représente le pourcentage de nœuds ayant un nœud père dans le réseau.

### 2.7.1 L'environnement de Simulation

Pour la validation de notre protocole *DiscoProto*, nous avons utilisé la plateforme de simulation Omnet++/Castalia [Ath ; Tse+10]. Ce simulateur implémente des modèles réalistes du canal radio et des modèles de propagation incluant le modèle *Lognormal Shadowing*. L'implémentation des nœuds de capteurs est quant à elle basée sur les spécifications réelles des capteurs CC2410 représentées sur les tableaux 2.7 et 2.8

| Spécifications | valeurs |
|---|---|
| Débit | 250 kbps |
| Type de modulation | PSK |
| Bande passante | 20 MHz |
| Bande passante bruit | 194 MHz |
| Niveau de Bruit | -100 dBm |
| Sensibilité | -95 dBm |
| Consommation d'énergie | 62 mW |

**Tab. 2.7.:** Spécifications en RX du coupleur radio de CC2410 implémentées dans les nœuds Castalia

| Tx (dBm) | 0 | -1 | -3 | -5 |
|---|---|---|---|---|
| Tx (mW) | 52.42 | 55.18 | 50.69 | 46.2 |

**Tab. 2.8.:** Spécifications des niveaux de Tx du coupleur radio de CC2410 implémentées dans les nœuds Castalia

Pour le modèle de propagation, nous avons choisi d'utiliser le modèle Lognormal Shadowing afin d'évaluer le comportement de notre protocole pour des environnement tels que les cours d'eau, les routes où des obstacles peuvent survenir durant la communication entre deux nœuds. Les paramètres utilisés pour le modèle de Shadowing sont : le taux d'affaiblissement du signal $\gamma = 2,4$ et la déviation Sigma $\sigma = 4$ de la composante aléatoire du modèle.

Le tableau 2.9 récapitule les paramètres de simulation pour ces différentes évaluations.

| | |
|---|---|
| Temps de simulation | 600, 1 000, 2 000, 3 000, 4000 (s) |
| Nombre de scénarios | 10 000 |
| Modèle de propagation | lognormal Shadowing ($\gamma = 2,4$, $\sigma = 4,0$) |
| Puissance de transmission | 0 dBm |
| Sensibilité | -95 dBm |
| Couche Physique | 802.15.4 |
| Couche MAC | TMAC |

**Tab. 2.9.:** Paramètres de simulation

Cependant, le simulateur Omnet++/Castalia implémente la couche MAC 802.15.4 en mode suivi de balise pour les réseaux en étoile qui permet seulement le routage à un saut. D'autres protocoles tels que SMAC [Ye+02] et TMACC [DL03] qui permettent le routage à plusieurs sauts sont aussi implémentés.

Face à l'inexistence, sur ce simulateur, du protocole 802.15.4 en mode non suivi de balise et qui permet faire du routage à plusieurs sauts, nous avons choisi le protocole TMAC pour les besoins de nos différentes évaluations. En effet, TMAC utilise le mode RTCS/CTS pour résoudre les problèmes de terminaux cachés et permet d'étendre si nécessaire les slots de temps d'émission des données afin d'éviter les phénomènes d'endormissement des nœuds.

Un exemple du fichier de paramètres utilisé dans l'environnement Castalia est présenté dans l'annexe E

### 2.7.2 Proximité avec la topologie physique

Les graphes de la figure 2.21 montrent que le nombre de branchements et le nombre de disjonctions découvertes sont linéairement corrélés. En effet, durant la formation du réseau, la plupart des disjonctions sont générées à l'endroit des branchements de la topologie physique et certaines rares disjonctions sont créées en plus dans le voisinage des branchements. Dans certains cas, nous remarquons que le nombre de disjonctions calculé pendant la formation de la topologie est plus petit que le nombre de branches de la topologie linéaire. Cela est dû au fait qu'avec *DiscoProto* certaines branches

parallèles et proches dans la topologie physique sont agrégées pour en former une seule dans la topologie logique.

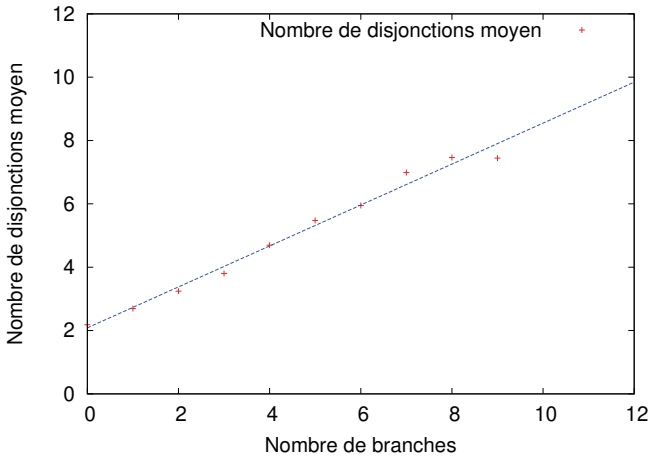
### 2.7.3 Influence du nombres de branchements sur le temps d'association moyen

Les graphes de la figure 2.22 montrent que le temps moyen d'association en fonction du nombre de disjonctions décroît jusqu'à une valeur dépendant de la taille du réseau. Ces résultats montrent que l'augmentation du nombre de branches ne crée pas de latence sur la formation du réseau. En effet, *DiscoProto* a l'avantage d'effectuer simultanément l'association de plusieurs nœuds sur des branches éloignées hors de portée radio l'une de l'autre lors de leur challenge respectif.

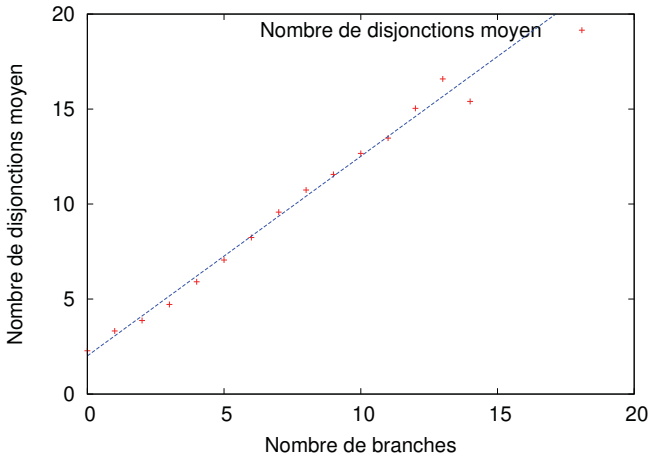
Cependant, pour des réseaux linéaires denses (comportant un grand nombre de branchements, un grand nombre de branches dans une même voisinage peut causer un allongement de la durée moyenne d'association parce qu'il augmente le nombre de nœuds participant à l'étape de challenge dans une même zone. Ce qui entraîne un rallongement du temps d'accès au canal radio entraînant un nombre élevé d'annulation d'associations et de ce fait augmente le nombre de cycles de collecte de fils. Pour les RdCSF linéaires longs ne comportant pas de branchements, le temps d'association est aussi très long car dépendant du nombre de nœuds. En effet, puisque un nœud est associé à chaque cycle, le temps moyen d'association d'un réseau de $N$ nœuds sera égal à l'équation (2.3) sous réserve qu'il n'y ait pas de collisions dans le réseau. Avec *DiscoProto* la durée théorique d'une association entre deux nœuds ($TempsAssoc$) est donnée par (2.4)

$$TempsMoyenReseau \simeq TempsAssoc \times N \tag{2.3}$$

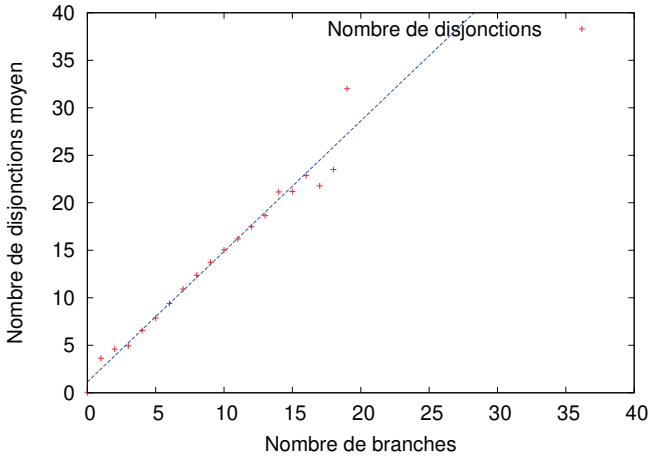
$$\begin{aligned} TempsAssoc_{Théorique} = \quad & Durée\_ColleteSonOffer + Durée\_Challenge \\ & + Durée\_Association \end{aligned} \tag{2.4}$$




**(a)** Réseaux de 50 nœuds

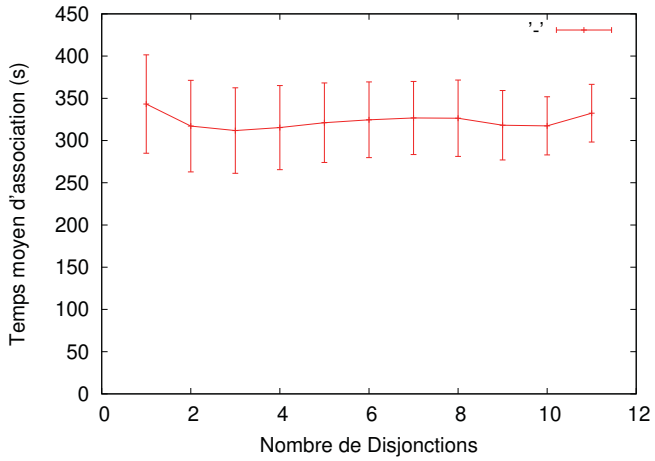



**(b)** Réseaux de 100 nœuds

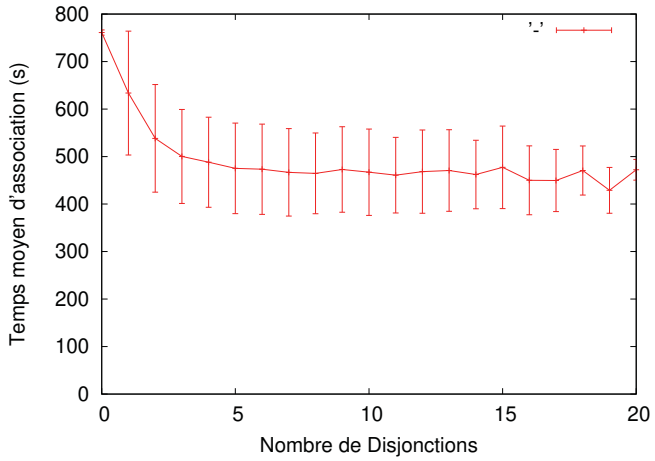



**(c)** Réseaux de 150 nœuds

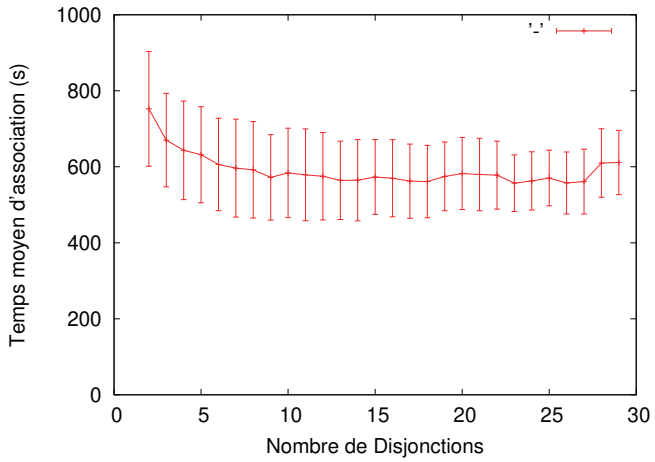
**Fig. 2.21.:** Nombre de disjonctions moyen par nombre de branches




**(a)** Réseaux de 50 nœuds




**(b)** Réseaux de 100 nœuds




**(c)** Réseaux de 150 nœuds

**Fig. 2.22.:** Temps moyen d'association par disjonction

### 2.7.4 Le passage à l'échelle

D'après les résultats des graphes de la figure 2.22 (résumés dans la figure 2.23), nous notons que *DiscoProto* supporte le passage à l'échelle et montre que la durée moyenne d'association n'augmente pas linéairement en fonction de la taille du réseau. En effet, comme mentionné dans la section 2.7.3, *DiscoProto* a la capacité d'effectuer parallèlement les collectes de nœuds le long des branches à condition qu'ils ne soient pas à portée de leur *Challenge* respectif. Ce qui fait que le temps d'association ne double pas lorsqu'on passe par exemple d'un réseau de 50 nœuds à un réseau de 100 nœuds.

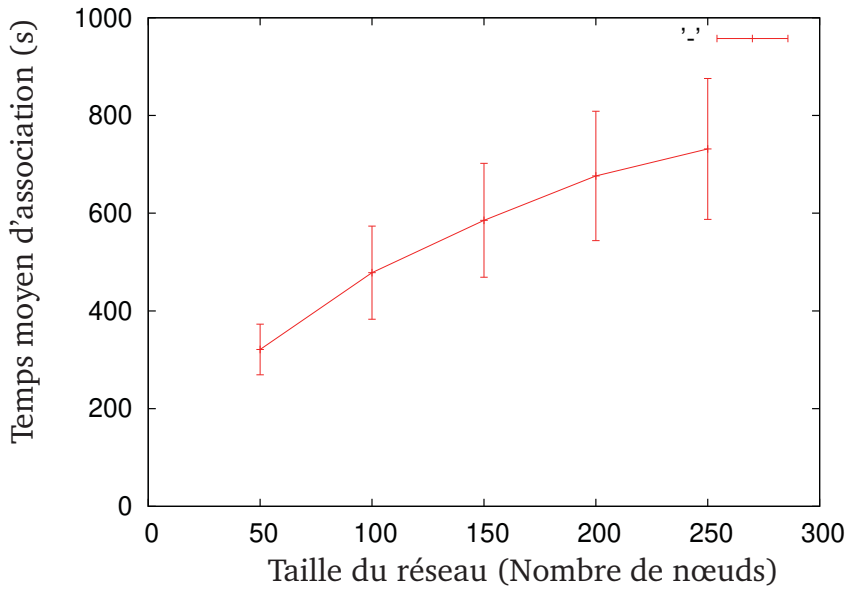



**Fig. 2.23.:** Temps moyen d'association par taille de réseau

### 2.7.5 L'efficacité du protocole

Pour évaluer l'efficacité de notre protocole à former une topologie de RdCSFL, nous avons mesuré le taux d'association de *DiscoProto* avec celui de DAAM ZigBee (Cm=2,Rm=2,Lm=15). La figure 2.24 montre que le taux d'association de DAAM de Zigbee décroît avec la taille du réseau car il atteint très rapidement les limites de configuration (Cm=2,Rm=2,Lm=15) définit à l'état initial du réseau. Cependant, le taux d'association de *DiscoProto* pour les différents réseaux avoisine les 100% causant ainsi l'apparition de quelques rares nœuds orphelins. Ce phénomène est causé par les problèmes de collisions qui surviennent dans certains scénarios de réseaux denses (réseaux

comportant beaucoup de branchements) et empêchent la réception des messages *FatherOffer* ou *SonOffer* limités à trois tentatives. Nous proposons dans la suite de ce manuscrit une solution pour associer plus tard ces nœuds orphelins en.

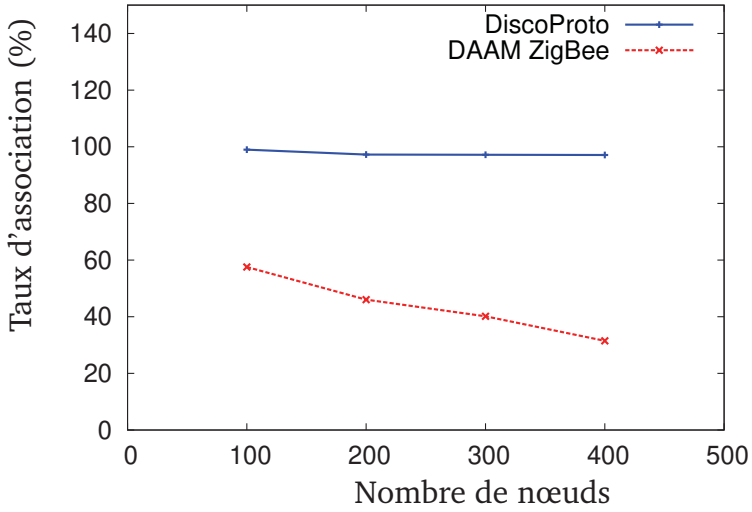



**Fig. 2.24.:** Taux d'association moyen par nombre de nœuds du réseau

### 2.7.6 Influence des temporisateurs sur la durée moyenne d'association

Le protocole *DiscoProto* fonctionne comme un algorithme glouton pour la formation de la topologie linéaire. Ceci implique l'association au moins d'un nœud à l'ensemble connecté à chaque tour de la phase de collecte de fils. Cette phase de collecte de fils repose pour son bon fonctionnement sur le paramétrage de certains temporisateurs énumérés dans le tableau ci-dessus (Tableau 2.10).

Le temporisateur NEIGHBOR_DISCOVERY_TIMEOUT est déclenché au démarrage de chaque nœud et leur permet de s'annoncer aux autres nœuds voisins grâce à l'utilisation du paquet *Hello* diffusé périodiquement. À l'expiration de ce temporisateur, chaque nœud aura dans sa table de voisinage l'ensemble des nœuds de son voisinage à un saut.

Le temporisateur COLLECTING_SONS_TIMEOUT détermine la durée d'un tour de collecte de fils pendant laquelle chaque nœud déjà connecté diffuse un message *FatherOffer*, reçoit d'éventuels messages *SonOffer* des nœuds

non connectés voisins pour choisir la meilleure offre $Objectif$. Si aucun message SonOffer n'est reçu, le nœud démarre une nouvelle tentative en déclenchant à nouveau le temporisateur dans la limite de trois tentatives.

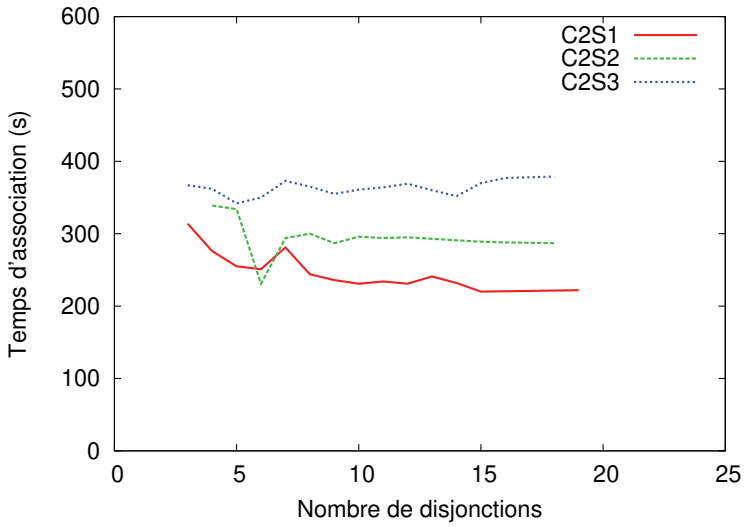
Le temporisateur CHALLENGE_TIMEOUT permet à un nœud qui reçoit au moins une offre $objectif$ d'effectuer sa phase de challenge afin de vérifier s'il existe dans son voisinage un nœud avec une meilleure offre que la sienne.

| Type de Temporisateur | Description | Phase |
|---|---|---|
| NEIGHBOR_DISCOVERY_TIMEOUT | Temporisateur pour la découverte de voisinage | Découverte de voisinage |
| COLLECTING_SONS_TIMEOUT | Temporisateur pour la collecte des offres (Objectif) envoyées par les nœuds ayant reçu un message FatherOffer | Collecte des nœuds fils |
| CHALLENGE_TIMEOUT | Délai pendant lequel un message ChallengeOffer et ChallengeOfferResponse sont diffusés pour élire le(s) nœuds qui doit(vent) continuer l'association | Collecte des nœuds fils |

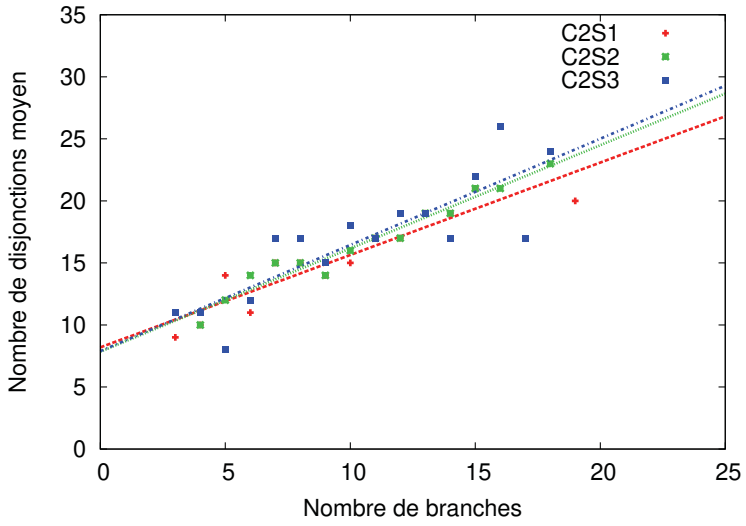
**Tab. 2.10.:** Différents temporisateurs de *DiscoProto*

Dans cette partie, nous étudions l'impact des valeurs des temporisateurs COLLECTING_SONS_TIMEOUT et CHALLENGE_TIMEOUT sur la durée moyenne d'un tour d'association, sur le taux d'association et sur la proximité de la topologie générée par rapport à la topologie physique. Pour ce faire, en se basant sur les paramètres de simulation définis dans le tableau 2.9, nous faisons varier les valeurs des temporisateurs COLLECTING_SONS_TIMEOUT et CHALLENGE_TIMEOUT avec les valeurs $1\ s$, $2\ s$, $3\ s$, $4\ s$.

Les résultats de nos évaluations de performance sont représentés dans les graphes ci-dessous. Dans ces différentes graphes, la notation $C2S3$ représente une durée de Challenge de $2\ secondes$ et une durée de Collecte des fils de $3\ secondes$. Dans les figures 2.25 et 2.26, nous fixons la valeur de la durée du challenge à $2$ et $3$ secondes puis nous faisons varier la durée de collecte des offres des nœuds fils entre $1$ et $3$ secondes. Les courbes montrent un gain de temps moyen d'association avec un petit temps de collecte des offres des nœuds fils (voir figures 2.25a et 2.26a). Cependant, ce temps



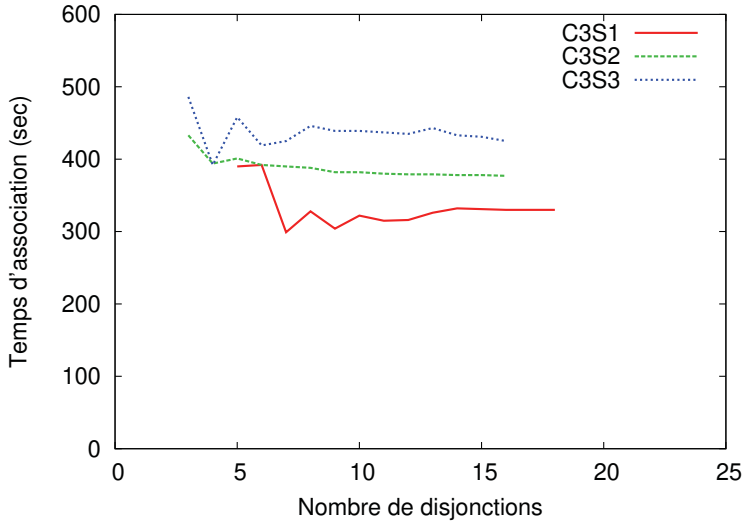

**(a)** Temps d'association moyen



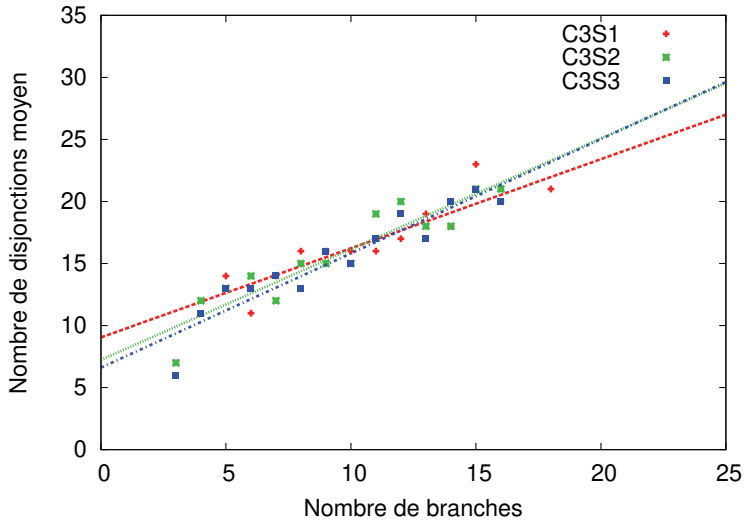

**(b)** Disjonctions par branchement

**Fig. 2.25.:** Temps moyen d'association pour $Durée\_Challenge = 2\ s$ et pour différentes durées de collecte de fils pour un réseau de $100$ nœuds

très petit de la durée de collecte ne permet pas de collecter toutes les offres devant être envoyées par les nœuds dans le voisinage d'un nœud connecté ayant diffusé un *FatherOffer*.



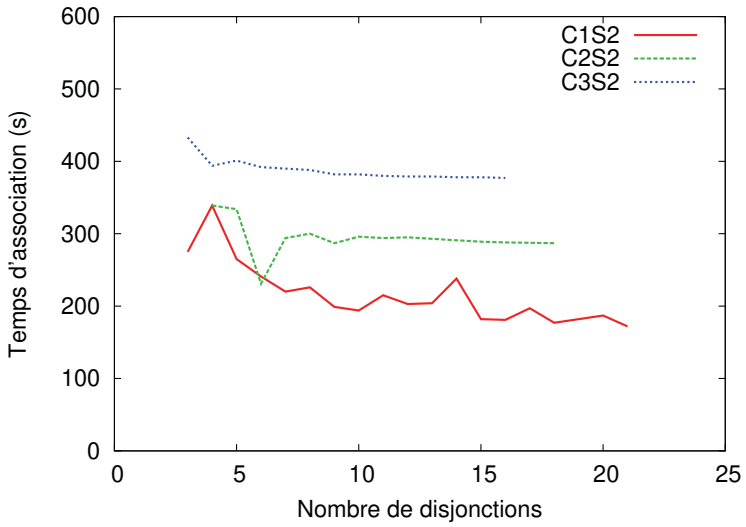

**(a)** Temps d'association moyen



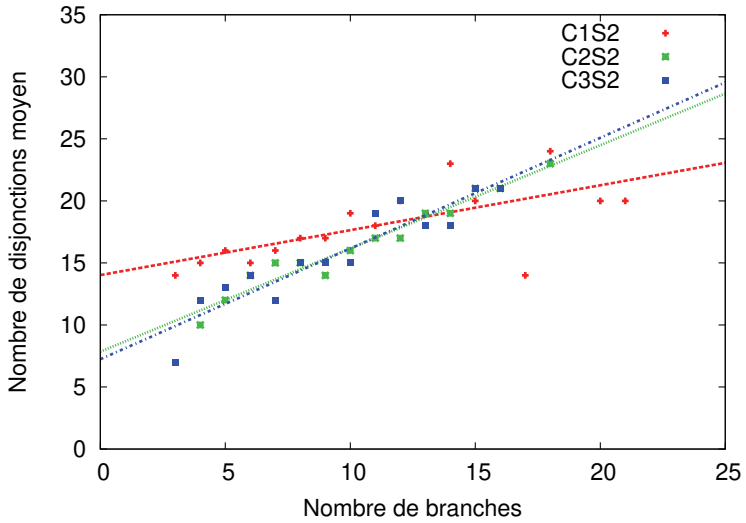

**(b)** Disjonctions par branchement

**Fig. 2.26.:** Temps moyen d'association pour une $Durée\_Challenge = 3\ s$ et pour différentes durées de collecte de fils pour un réseau de $100$ nœuds

De ce fait, la topologie qui sera construite aura un rapport Disjonction/-Branchement très grand (Voir figures 2.25b et 2.26b) parce que beaucoup de nœuds vont connecter plusieurs nœuds fils, car n'ayant pas le temps de collecter toutes les offres dans leur voisinage.




**(a)** Temps d'association moyen




**(b)** Disjonctions par branchement

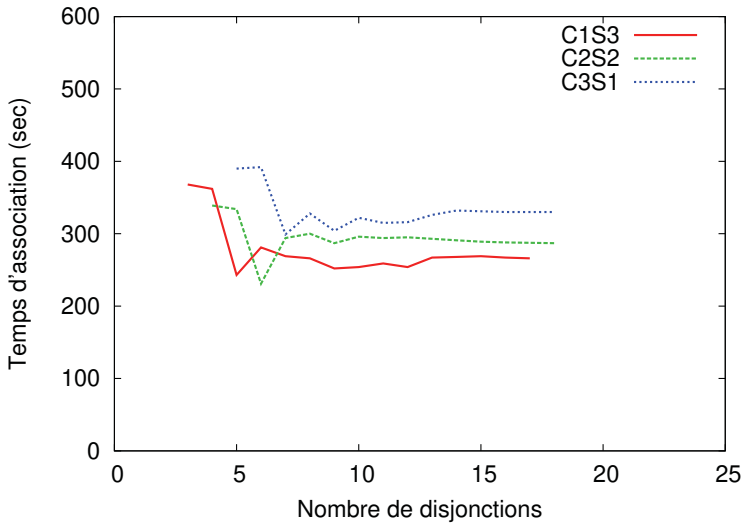
**Fig. 2.27.:** Temps moyen d'association pour $Durée\_Collecte = 2\ s$ et pour différentes durées de challenge pour un réseau de 100 nœuds

Dans la figure 2.27, nous fixons la valeur de la durée de collecte à 2 secondes puis nous faisons varier la durée du challenge entre 1 et 3 secondes.

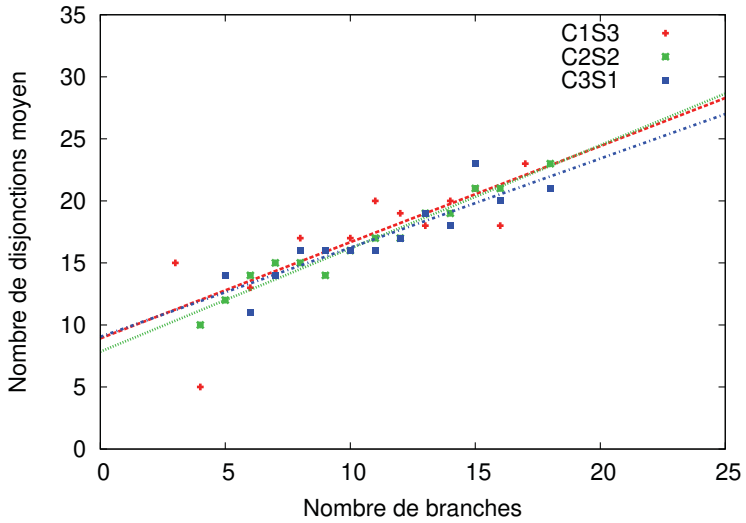
Les courbes montrent que même si le fait de diminuer la durée de challenge offre un temps moyen d'association plus petit, il induit l'apparition de plusieurs disjonction sur un segment linéaire.

La figure 2.28 confirme nos résultats précédents et montre qu'avec une durée de collecte des offres des fils et une durée de challenge très petites, cela ne favorise pas la construction d'une topologie proche de la topologie physique car beaucoup de nœuds généreront trop de nœuds fils.

En résumé, nous pouvons dire que la durée de collecte de fils et la durée de challenge sont des étapes très importantes dans la phase de découverte et de construction de la topologie et le meilleur compromis pour avoir une durée optimale d'association dépend de la densité des nœuds dans le voisinage. En effet, pour des RdCSFL denses avec beaucoup de branchements, il serait judicieux de choisir des temps de collecte de fils et de challenge supérieurs à deux secondes chacune afin de prendre en compte les retransmissions des messages dues aux collusions et permettre au nœuds d'un même voisinage de recevoir et répondre aux différents messages *FatherOffer* reçus. Pour des RdCSFL moins denses, des temps de collecte de fils et de challenge de deux secondes permettant de réaliser des cycles dans chaque voisinage du réseau.




**(a)** Temps d'association moyen




**(b)** Disjonction par branchement

**Fig. 2.28.:** Temps moyen d'association pour une durée $Challenge + Collecte = 4\ s$ pour un réseau de 100 nœuds

## 2.8 Conclusion

Dans cette partie, nous avons présenté notre protocole de découverte et de formation de RdCSF Linéaire *DiscoProto* qui permet, à partir d'un ensemble de nœuds de capteurs déployés le long d'un environnement linéaire, de créer une topologie linéaire de RdCSF. *DiscoProto* forme cette topologie sans connaissance de la profondeur, sans limitation du nombre de nœuds fils par nœuds et sans connaissance de la position des nœuds. Il fournit un mécanisme d'adressage en se basant uniquement sur la taille réelle du réseau découvert et une répartition uniforme des adresses à destination de chaque nœud. Nous avons d'abord défini les réseaux de capteurs sans fil linéaires de façon générale. Pour ce faire, nous avons introduit la notion de facteur de linéarité basée sur le pourcentage de nœuds simples dans un réseau. Nous avons ensuite présenté notre approche centralisée de *DiscoProto*. Enfin, nous avons décrit le fonctionnement de la version distribuée de *DiscoProto* implémentée sur l'environnement de simulation Castalia.

Grâce au facteur de linéarité et au coefficient de redondance, nous avons filtré les topologies linéaires formées par notre générateur afin de les utiliser pour évaluer notre protocole *DiscoProto*. Les résultats des simulations réalisées sur Castalia montrent que notre protocole supporte le passage à l'échelle et est adapté à tout type de réseaux linéaires, réussissant ainsi à former la topologie linéaire avec un taux d'association avoisinant les $100\%$. Le déploiement $2$-redondants des nœuds offre une plus grande résistance face à la défaillance d'un lien. En effet, il permet à un nœud de capteurs qui a perdu un nœud père de transférer les données à son nœud grand père (en direction du puits).

Le choix des bonnes valeurs pour les temporisateurs de la durée de collecte des offres de nœuds fils et de la durée du challenge permet de former une topologie avec moins d'apparition de disjonctions le long des différents segments linéaires et d'avoir une topologie se rapprochant de la topologie physique tout en optimisant la durée moyenne d'association.

*DiscoProto* ne permet pas d'étendre le réseau formé pour l'intégration de nouvelles branches de nœuds ou bien de nœuds éparses lorsque nous voulons, par exemple, surveiller un nouveau tronçon d'autoroutes, de voies ferrées

ou de cours d'eau. Il laisse aussi parfois quelques nœuds orphelins car sans possibilité de rejoindre le réseau à la fin de la phase de collecte des fils. Notre deuxième contribution propose un nouveau protocole appelé *Dynamic DiscoProto*. *Dynamic DiscoProto* va permettre l'extension d'un réseau de capteurs sans fil linéaire avec l'ajout de nouveaux nœuds et la gestion des réallocations de nouveaux blocs d'adresses pour les nœuds en cas d'épuisement de leur réserve d'adresses $Fskip$ ou lorsque ce dernier est insuffisant pour adresser l'ensemble des nœuds d'une nouvelle branche ajoutée au RdCSF linéaire. *Dynamic DiscoProto* va assurer aussi l'ajout de puits secondaires et va implémenter un mécanisme de routage permettant l'acheminement des données à destination de ces puits.

# 3

# Mécanismes de gestion d'un RdCSF Linéaire dynamique

Le protocole *DiscoProto* est limité par le fait qu'il ne prend pas en charge les réseaux de capteurs linéaires dynamiques où l'ajout et la suppression de nœuds de capteurs sont possibles après la construction du réseau. De plus, *DiscoProto* ne prend en charge que le routage des données à destination d'un seul puits.

Dans [Par+09], les auteurs ont proposé un mécanisme appelé *DIBA* (DIstributed Borrowing Addressing ) permettant d'ajouter de nouveaux nœuds et d'emprunter des adresses à d'autres nœuds dans un réseau de capteurs sans-fil ZigBee pour une topologie en arbre. *DIBA* permet à un nœud ne pouvant plus associer de nouveaux nœuds parce qu'ayant atteint ses limites de configurations définies par $C_m$,$R_m$,$L_m$ (voir la section 1.5 à la page 48), d'emprunter des blocs d'adresses disponibles sur d'autres nœuds. Cependant, pour des réseaux de capteurs sans-fil dynamiques avec des paramètres $Cm$,$Rm$,$Lm$ mal dimensionnés, ce mécanisme *DIBA* entraine l'apparition de tables de routage complexes qui rendent difficile le routage des données vers leur destination.

Dans cette deuxième partie de notre contribution, nous proposons un nouveau protocole appelé *Dynamic DiscoProto* basé sur *DiscoProto* et permettant l'extension d'un RdCSF Linéaire. Par exemple, lorsque nous voulons surveiller de nouvelles portions d'autoroutes, d'affluents d'un fleuve ou d'une canalisation de gazoduc ajoutés à une infrastructure existante, *Dynamic DiscoProto* va fournir toutes les fonctionnalités permettant l'intégration de nouveaux nœuds de capteurs tout en assurant la continuité des services captage et le routage des données jusqu'au puits. De ce fait, *Dynamic DiscoProto* intègre toutes les fonctionnalités de découverte de topologie et d'adressage de *DiscoProto*. De plus, il prend en charge les fonctionnalité ci-dessous :

— L'ajout de nouveaux nœuds épars ou de nœuds formant une branche,

— La réallocation d'adresses par le nœud puits lorsque le bloc d'adresses $Fskip$ qui a été réservé sur un nœud est épuisé ou ne suffit plus pour l'adressage de nouveaux nœuds. Cette réallocation doit se réaliser tout en minimisant les échanges de messages de contrôle et la complexité des tables de routage.

Dans les RdCSF linéaires, les paquets de données sont relayés de nœuds en nœud jusqu'au le puits. Cette caractéristique peut engendrer une latence pendant la livraison des paquets de données lorsque le RdCSF linéaire a une grande profondeur (nombre de sauts entre le puits et le nœud d'extrémité ou de périphérie). Cette technique de routage des données dans les RdCSF linéaires engendre aussi une augmentation du trafic à chaque saut à destination du puits car chaque nœud relaie ses propres données capturées et les données reçues de ses nœuds fils l'ayant utilisé comme relais.

Soit $DA$ le débit de données applicatives généré par chaque nœud d'un réseau de $N$ nœuds, le débit de donnée à transmettre $DT$ par chaque nœud $i$ du chemin à destination du puits est donné par l'équation (3.1)

$$DT(i) = DA(i) + \sum_{fils=0}^{n} DT(fils) \tag{3.1}$$

Cette augmentation de trafic occasionne, dans le cas d'un RdCSF linéaire long (nombre de sauts élevés) ou comportant un grand nombre de branchements, des encombrements sur les nœuds au voisinage du puits ainsi que des débordements de files d'attente et des collisions. Pour résoudre ces problèmes, *Dynamic DiscoProto* permet, grâce à sa fonctionnalité d'ajout de nouveaux nœuds, d'ajouter des nœuds puits secondaires au niveau de certains endroits du réseau. Afin d'assurer l'acheminement des données collectées par les nœuds vers les puits du réseau, *Dynamic DiscoProto* implémente une fonctionnalité de routage multi-puits.

Pour la mise en œuvre de *Dynamic DiscoProto*, nous considérons un RdCSFL connecté, où chaque nœud a un nœud père et a obtenu son adresse logique ainsi que son bloc de réserve d'adresses $Fskip$ pour l'ajout d'éventuels nouveaux nœuds.

## 3.1 Ajout de nouveaux nœuds

Pour la mise en œuvre de ce mécanisme d'ajout de nouveaux nœuds, *Dynamic DiscoProto* utilise le même principe de découverte et de choix du meilleur lien d'association dans un voisinage basé sur la valeur du coût du lien appelé $Objectif$ décrit dans la section 2.3 à la page 78.

Avec *Dynamic DiscoProto*, deux types d'extension du RdCSFL sont possibles : l'ajout de nœuds épars disséminés à différents endroits du réseau et l'ajout de branches contenant des nœuds organisés linéairement.

Pour chacune de ces deux approches, les nouveaux nœuds de capteur ainsi que leurs nœuds voisins déjà connectés entame une phase de découverte de voisinage. Cette phase permet à chaque nœud de connaître ses nœuds voisins et de mettre à jour sa table de voisinage. Pour ce faire, dès que les nouveaux nœuds sont démarrés, ils commencent par diffuser un message *HELLO* dans leur voisinage. Chaque nœud connecté qui reçoit des messages *HELLO* diffuse à son tour un message *HELLO* et arme un temporisateur. À la fin de son temporisateur, chaque nœud connecté démarre la phase de collecte de nouveaux nœuds. Cette phase de collecte de fils suit les mêmes étapes que dans le protocole *DiscoProto* décrit dans la partie 2.4.

À la fin de la phase de collecte de nouveaux nœuds, chaque nœud connecté attribue les adresses à partir de son bloc de réserve d'adresses restantes et en fonction du nombre de nouveaux nœuds collectés. Lorsque le nombre de nouveaux nœuds est supérieur au bloc de réserve d'adresses restantes du nœud connecté, celui-ci entame une procédure de ré-allocation d'adresses afin de solliciter un nouveau bloc d'adresses au PAN coordinateur.

Nous présentons ci-dessous en détail les différentes étapes d'ajout dans le cas de nouveaux nœuds épars et dans le cas de branches de nœuds.

### 3.1.1 Cas de nouveaux nœuds épars

Dans le cas de l'ajout de plusieurs nœuds épars, chaque nouveau nœud commence par diffuser un message *HELLO* (voir figure 3.1a). Chaque nœud déjà

connecté qui reçoit ce message diffuse à son tour un *HELLO* et initialise un temporisateur pour le démarrage d'une nouvelle phase de collecte de nouveaux nœuds fils (voir figure 3.1b). Cette procédure permet aux nouveaux nœuds et à leur voisinage de mettre à jour leur table de voisinage.

À l'expiration du temporisateur, les nœuds connectés initient la phase de collecte de fils en diffusant un message *FatherOffer* incluant le nombre de nœuds fils déjà associés et le nombre de voisins (voir figure 3.1c). Chaque nouveau nœud, qui n'est pas encore connecté et qui a reçu un message *FatherOffer* répond par un message *SonOffer* incluant la valeur de l'$Objectif$ (voir figure 3.1d).

Chaque nœud connecté collecte ses messages *SonOffer* et choisit celui avec la meilleure valeur de l'$objectif$. À la fin du temporisateur de collecte des *SonOffer*, les nœuds connectés qui étaient en phase de collecte de fils diffusent un message *ChallengeOffer* contenant la valeur de l'$Objectif$ choisie. Puisque, plusieurs nœuds connectés peuvent être en compétition pour associer un même nouveau nœud par exemple. La phase de Challenge permet de choisir le meilleur lien d'association entre le nouveau nœud et ses voisins connectés. Le nœud connecté qui gagne le challenge dans un même voisinage continue la phase d'association du nœud. Ce nœud nouvellement associé va commencer une phase de collecte à son tour. Chaque nœud qui termine sa phase de collecte envoie à son nouveau nœud père le nombre de nœuds de son sous arbre incrémenté de $1$ inclus dans un message *PropaSon*. Si le nouveau nœud épars est le seul nœud dans son voisinage, il remonte la valeur $1$ à son nouveau nœud père. Lorsque le nœud père reçoit le message *PropaSon*, il attribue au nœud émetteur la première adresse parmi son bloc d'adresses disponibles (Figure 3.1f). Dès que le nouveau nœud reçoit son adresse, il peut démarrer le captage et l'envoi des données au puits

La figure C.2 de l'annexe C montrent des exemples de topologies formées après l'ajout de nouveaux nœuds épars avec le protocole *Dynamic DiscoProto*.

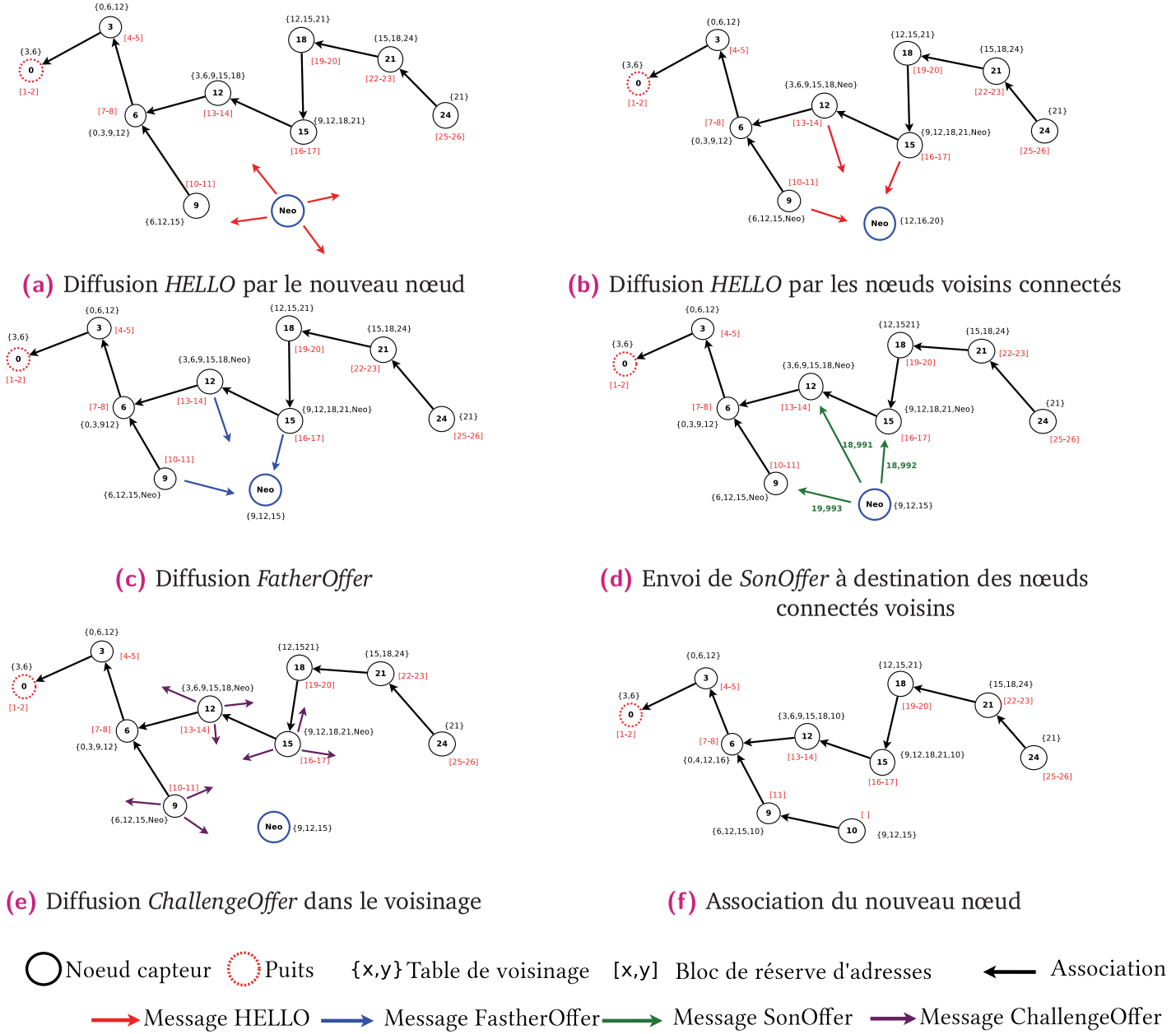



**Fig. 3.1.:** Procédure d'ajout de nouveaux nœuds épars

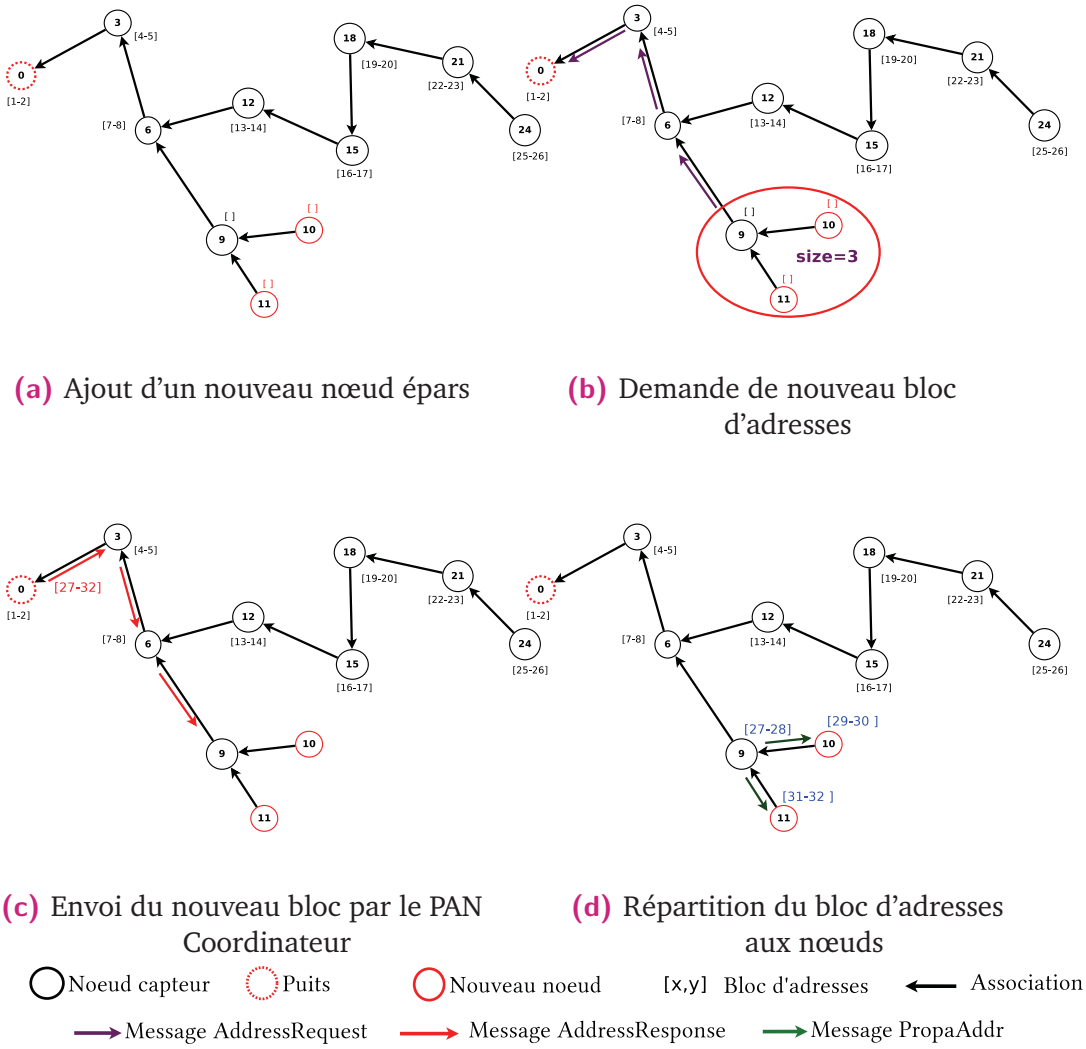



**Fig. 3.2.:** Procédure de réallocation d'adresses dans le cas d'ajout de nouveaux nœuds épars

Cependant, cette procédure d'ajout de nouveaux nœuds épars peut être utilisée dans le cas d'une nouvelle branche de ligne déjà constituée. Dans ce cas, la répétition de la phase d'ajout autant de fois que de nœuds dans la branche va engendrer beaucoup de messages de contrôles, et des procédures d'ajout plus longues. À cela s'ajoute un risque de dispersion des adresses surtout si de nouvelles branches se présentent simultanément pour être ajoutées.

Par conséquent, nous proposons, dans le cas de l'ajout de nouvelles branches de nœuds une autre procédure permettant l'association des nœuds en une seule phase de collecte.

### 3.1.2 Cas de nouvelles branches de nœuds

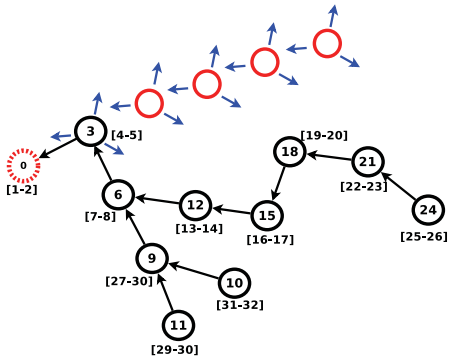
L'ajout de nouvelles branches de nœuds suit la même procédure que l'ajout de nœuds épars. Cependant, *Dynamic DiscoProto* commence par associer le nœud de la nouvelle branche ayant la meilleure offre (*Objectif*) avec les nœuds connectés de son voisinage. Ensuite, chaque nœud nouvellement associé de la branche, mais encore sans adresse logique, va démarrer, à son tour, sa phase de collecte de fils et sera en compétition avec les autres nœuds déjà connectés de son voisinage. Le dernier nœud de la branche à être associé va démarrer la phase de remontée de la taille du sous-arbre jusqu'au nœud connecté qui a associé le premier nœud de la nouvelle branche.

Grâce à cette approche de collecte de tous les nœuds fils avant d'avoir reçu les adresses, une seule demande d'adresses est effectuée pour une nouvelle branche. Non seulement cela réduit le nombre de messages de contrôle mais aussi cela permet d'être sûr de n'ajouter au plus qu'une entrée dans les tables de routage, au lieu d'ajouter potentiellement autant d'entrées que de nœuds dans la nouvelle branche.

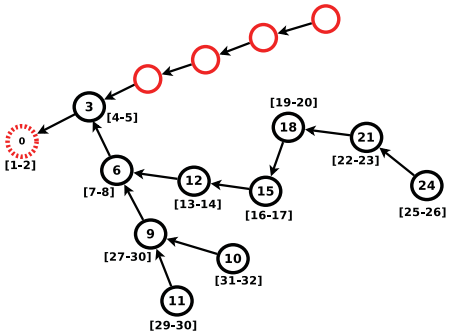
Si la taille de la nouvelle branche remontée à ce nœud connecté est inférieure au bloc d'adresses disponible alors dans ce cas le nœud initie la phase de distribution d'adresses en calculant le bloc d'adresses nécessaire pour la nouvelle branche en fonction de la taille reçue. Dans le cas échéant, le nœud lance une procédure de réallocation d'adresses envoyée au nœud puits.

Sur la figure 3.3d, le nœud $3$ reçoit, de la nouvelle branche la taille $4$ alors qu'il n'a que deux adresses disponibles. Dans ce cas, il démarre la procédure de réallocation d'adresses (détaillée dans la section suivante) en émettant à destination du puits une requête *AddressRequest* incluant la taille de la branche de nœuds à pourvoir en adresses. À la réception du nouveau bloc d'adresses (Figure 3.3e), le nœud $3$ le distribue aux nœuds de la nouvelle branche.

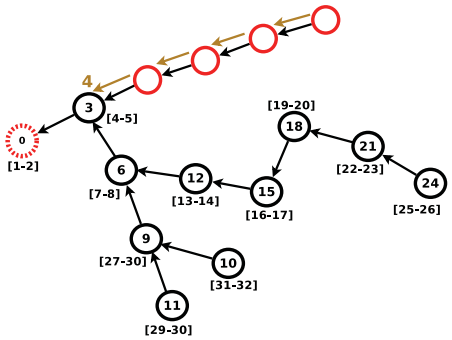
La figure C.3 de l'annexe C montrent des exemples de topologies formées après l'ajout de nouvelles branches avec le protocole *Dynamic DiscoProto*.



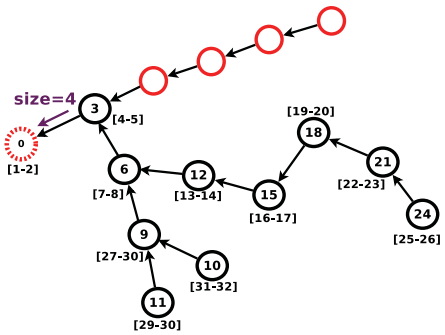

(a) Découverte de voisinage



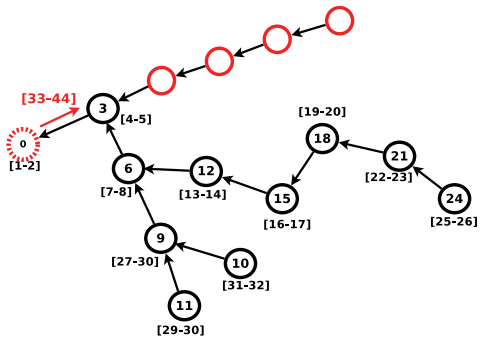

(b) Association de la branche de nœuds



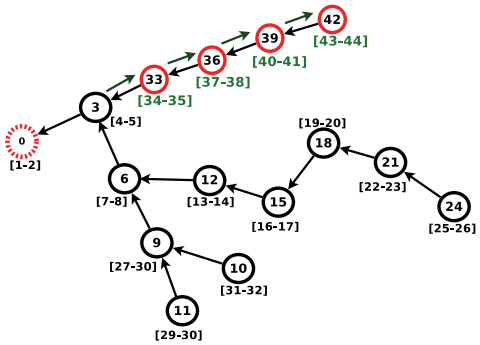

(c) Collecte de la taille de la branche




(d) Demande d'un nouveau bloc d'adresses




(e) Réception du nouveau bloc d'adresses




(f) Répartion du bloc d'adresses aux nœuds de la branche

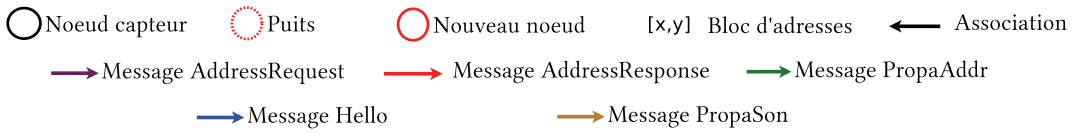



**Fig. 3.3.:** Procédure d'ajout d'une nouvelle branche de nœuds

**Algorithm 3.1:** Traitement à la réception d'un message **PropaSize**

**Result:** Réaction suite à l'ajout de nouveaux nœuds.
**Input** : $SizeNewSubtree$ Réception de la taille d'un sous-arbre venant d'un nouveau nœud ou d'une nouvelle branche
**Input** : $SonMacaddr$ l'adresse MAC du nouveau nœud
**Input** : $nn$ Nombre de nouveaux nœuds ajoutés

```
1  if isConnectedwithAddr = true then
       // Le nœud est associé et à une adresse
2      if Fskip ≥ SizeNewSubtree then
           // Assez d'addresses disponibles
3          nn ← nn + SizeNewSubtree ;
4          Fskip ← Fskip − SizeNewSubtree ;
5          StartBlock ← NodeAddr + 1;
6          EndBlock ← NodeAddr + SizeNewSubtree ;
7          NewNodeFskip ← 0;
8          PropaAddr (SonMacaddr, StartBlock, EndBlock, NewNodeFskip) ;
9          Unicast (SonMacaddr, PropaAddr);
10         if Fskip = 0 then
               // Demande de bloc d'addresses pour les Fskip
11             AddrRequest (NodeAddr, nn) ;
12             Unicast (SinkAddr, AddrRequest);
13     else
           // Demande de bloc d'addresses pour les nouveaux nœuds
              avec Fskip
14         AddrRequest (NodeAddr, SizeNewSubtree) ;
15         Unicast (SinkAddr, AddrRequest);
```

## 3.2 Réallocation d'adresses

Lors de la phase d'ajout de nouveaux nœuds épars ou de nouvelles branches de nœuds, deux situations peuvent se produire.

— Le nœud déjà connecté a épuisé son bloc de réserve d'adresses $Fskip$ et n'est plus en mesure de fournir des adresses aux nouveaux nœuds épars ajoutés ,

— Le nœud déjà connecté a ajouté une nouvelle branche de nœuds de telle sorte que la taille de cette dernière est supérieure au bloc de réserve d'adresses disponibles $Fskip$

Dans ces deux cas, le nœud entame une procédure de réallocation d'adresses. Un message *AddressRequest* est envoyé au nœud puits du réseau qui va calcu-

ler un nouveau bloc d'adresses proportionnellement à la taille des nouveaux nœuds à adresser.

### 3.2.1 Cas de nouveaux nœuds épars

Lorsqu'un nœud du réseau a ajouté de nouveaux nœuds épars jusqu'à épuisement de son bloc de réserve d'adresses $Fskip$, il émet à destination du puits une requête *AddressRequest* contenant la valeur $Taille = Fskip + 1$ qui correspond au nombre de nœuds nouvellement ajoutés et du nœud à l'origine de la requête. Cette valeur permettra au puits de calculer un nouveau bloc d'adresses à distribuer aux nouveaux nœuds mais aussi au nœud qui a épuisé son bloc d'adresses $Fskip$ afin qu'il puisse ajouter d'autres nouveaux nœuds. À la réception de la requête *AddressRequest*, le puits calcule la taille du nouveau bloc d'adresses grâce à la formule (3.2).

$$TailleNewBloc = [oldBlock + 1, oldBlock + Fskip \times taille \tag{3.2}$$

Le figure 3.2 montre les étapes de la procédure de réallocation d'adresses dans le cas des nœuds épars. Dans cet exemple, le nœud $9$ a ajouté successivement deux nouveaux nœuds à qui il a attribué les deux adresses de son bloc à savoir $10$ et $11$ (voir figure 3.2a). Après avoir épuisé son bloc d'adresses, le nœud $9$ transmet une requête *AddressRequest* au puits avec comme valeur de $Taille = 3$ pour solliciter un nouveau bloc d'adresses pour lui et pour ses nouveaux nœuds ajoutés. À la réception de la requête, le puits calcule le nouveau bloc en se basant sur le dernier bloc d'adresses distribué dans le réseau. Dans la figure 3.2c, le PAN coordinateur avait déjà distribué le bloc d'adresses $[0 - 26]$, de ce fait il calcule le nouveau bloc d'adresses $[27 - 32]$ et l'inclut dans le message *AddressResponse* à destination du nœud $9$.

À la réception du nouveau bloc d'adresses, le nœud $9$ garde le bloc $[27 - 28]$ comme réserve d'adresses $Fskip$ et fait la répartition du bloc restant à l'ensemble des nouveaux nœuds (bloc $[29 - 30]$ au nœud $10$ et bloc $[31 - 32]$ au nœud $11$).

### 3.2.2 Cas de nouvelles branches de nœuds

Lorsqu'un nœud du réseau ajoute une nouvelle branche de nœuds et que la taille de cette dernière est supérieure au bloc de réserve d'adresses disponibles restant, alors le nœud fait une demande de réallocation de nouveau bloc d'adresses en émettant une requête *AddressRequest* contenant la valeur $Taille = Size(NewSubTree)$. Le puits calcule la taille du nouveau bloc d'adresses grâce à la formule (3.3) avant de répondre par un paquet *AddressResponse*.

$$TailleNewBloc = [oldBlock + 1, oldBlock + (Fskip + 1) \times taille] \quad (3.3)$$

À la réception de ce paquet, le nœud à l'origine de la requête distribue le bloc d'adresses proportionnellement au besoin de chaque nœud de la branche.

Dans la figure 3.3c, le nœud $3$ vient d'ajouter une branche contenant $4$ nœuds alors qu'il a deux adresses disponibles (valeur par défaut de $Fskip$). Dans ce cas, le nœud $3$ initie une demande ré-allocation d'adresses à destination du puits pour $4$ nœuds (voir figure 3.3d).

Le puits, à la réception de la requête du nœud $3$, regarde dans sa table d'allocation d'adresses et calcule le nouveau bloc d'adresse pour le nœud $3$. Dans la figure 3.3e, le puits, ayant déjà attribué le bloc d'adresses $[0 - 32]$ aux nœuds du réseau, attribue le bloc d'adresses $[33 - 44]$ au nœud $3$ pour l'adressage de la nouvelle branche. À la réception du nouveau bloc, le nœud le distribue aux nœuds de la branche en utilisant le message *PropaAddr* (voir figure 3.3f).

## 3.3 Gestion des tables d'adresses

Pour assurer le bon fonctionnement du routage des données à destination du puits du réseau, le protocole *Dynamic DiscoProto* implémente une fonctionnalité de gestion des adresses allouées aux nouveaux nœuds. *Dynamic DiscoProto* maintient une table de routage ayant le même format que celle de *DiscoProto*. Pour ce faire, lors de l'envoi des nouveaux blocs d'adresses par le puits, chaque nœud, traversé par le message *AddressResponse* conte-

nant le nouveau bloc d'adresses, ajoute une nouvelle entrée dans sa table de routage avant de relayer le message au nœud prochain saut. Cette procédure permet de connaître dans quelle sous-branche se trouve ce nouveau bloc et de maintenir le routage des données à destination de chaque nœud du réseau.

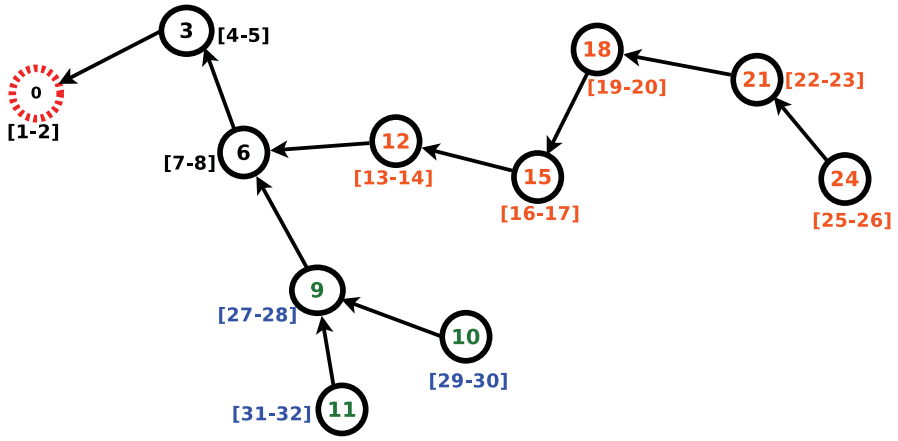



**Fig. 3.4.:** Réseau après ajout des nouveaux nœuds épars

| start_Block | end_Block | addr_Node | id_Son |
|---|---|---|---|
| 6 | 26 | 6 | 2 |
| 27 | 32 | 6 | 2 |

**Tab. 3.1.:** Table de routage du nœud 3 de la figure 3.4

| start_Block | end_Block | addr_Node | id_Son |
|---|---|---|---|
| 9 | 11 | 9 | 3 |
| 12 | 26 | 12 | 4 |
| 27 | 32 | 9 | 3 |

**Tab. 3.2.:** Table de routage du nœud 6 de la figure 3.4

Dans l'exemple de la figure 3.4, le nœud 3 avait déjà dans sa table l'entrée du bloc d'adresses $[6-26]$ à destination du nœud 6. Lorsque le message *AddressResponse* contenant le nouveau bloc d'adresses $[27-32]$ envoyé par le PAN coordinateur traverse le nœud 3 à destination du nœud 9, alors ce dernier ajoute la nouvelle entrée $[27-32]$ avec comme prochain saut le nœud 6 (voir table 3.2).

Le nœud $6$ quant à lui, avait deux entrées dans sa table à savoir l'entrée $[9-11]$ à destination du nœud $9$ et l'entrée $[12-26]$ à destination du nœud $12$. À la réception du message *AddressResponse* contenant le nouveau bloc d'adresses $[27-32]$, le nœud $6$ ajoute à son tour la nouvelle entrée $[27-32]$ avec comme destination le nœud $9$ (voir table 3.2).

Dans l'exemple de la figure 3.5, lorsque le nœud $3$ reçoit à nouveau un message *AddressResponse* contenant un nouveau bloc d'adresses $[33-44]$ pour les nœuds de la nouvelle branche qu'il vient d'ajouter, alors il ajoute la nouvelle entrée $[33-44]$ dans sa table de routage avec comme nœud de destination le nœud $33$ (voir table 3.3).

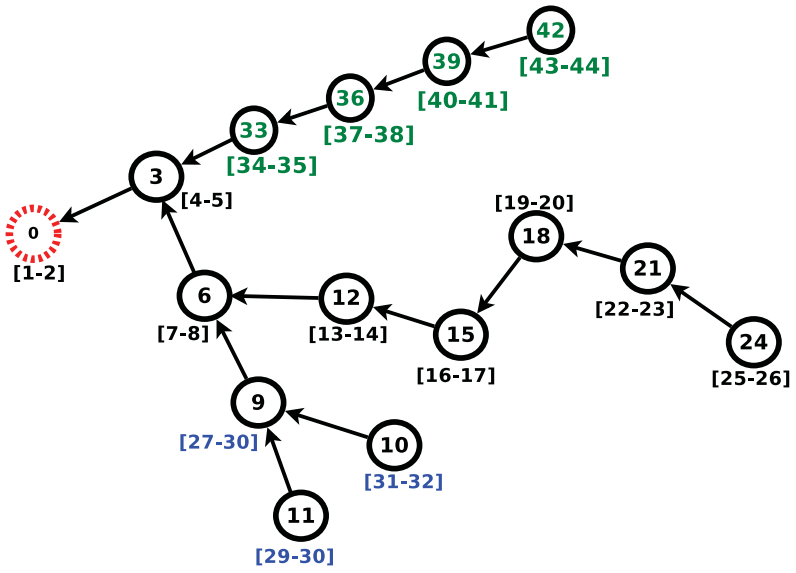



**Fig. 3.5.:** Réseau après ajout d'une nouvelle branche de nœuds

| start_Block | end_Block | addr_Node | id_Son |
|---|---|---|---|
| 6 | 26 | 6 | 2 |
| 27 | 32 | 9 | 2 |
| 33 | 44 | 33 | 11 |

**Tab. 3.3.:** Table de routage du nœud $3$ de la Figure 3.5

Durant la phase de réallocations d'adresses, il peut arriver qu'un nœud donné ait dans sa table des entrées de blocs d'adresses consécutives destinées à un même nœud. Dans ce cas, il est possible de faire la fusion de ces entrées afin d'optimiser l'espace mémoire occupé et le temps nécessaire à l'exécu-

tion du routage des données au puits. Dans l'exemple de la figure 3.4, la table contient deux entrées consécutives $[6, 26]$ et $[27, 32]$ destiné au même nœud $6$. Après la fusion de ces deux entrées (figure 3.5), nous avons plus qu'une seule entrée $[6, 32]$ pour le nœud $6$.

| start_Block | end_Block | addr_Node | id_Son |
|---|---|---|---|
| 6 | 26 | 6 | 2 |
| 27 | 32 | 6 | 2 |

**Tab. 3.4.:** Table de routage du nœud 3 de la figure 3.4

| start_Block | end_Block | addr_Node | id_Son |
|---|---|---|---|
| 6 | 32 | 6 | 2 |

**Tab. 3.5.:** Table de routage fusionnée du nœud 3 de la figure 3.4

## 3.4 Routage multi-puits dans les RdCSF linéaire

L'un des spécificités des RdCSFL concerne le drainage des paquets de données à destination du puits de collecte. En effet, dans un tel réseau, les données sont relayés sur de longues distances (nombre de sauts élevés) ce qui peut augmenter le délai de bout en bout mais aussi les risques de perte durant le transit. De plus, on note aussi une augmentation du débit de données au fur et à mesure qu'on se rapproche du puits occasionnant de fortes congestions, des risques de débordement de files d'attente et de collisions élevés. Ceci peut impacter négativement sur le ration de paquets reçus par le puits. Dans cette partie nous décrivons une fonctionnalité de *Dynamic DiscoProto* permettant la gestion du routage vers plusieurs puits d'un réseau de capteur sans fils linéaire.

*Dynamic DiscoProto* permet d'ajouter des nœuds jouant le rôle de puits pour la collecte des données transmises par les nœuds du réseau en suivant la même procédure d'ajout de nœuds épars décrite dans la section 3.1.1. Ces

nœuds puits seront appelés *puits secondaires* à la différence du *puits principal* qui est le coordinateur du réseau et qui se charge de l'adressage et de la réallocation des adresses des nœuds mais aussi de la collecte de données transmises par les nœuds. De ce fait, le gestionnaire du réseau de capteurs pourra déployer des puits secondaires afin de réduire le nombre de sauts pour atteindre le puits principal ou de réduire les zones d'encombrement et de congestion. Pour permettre à chaque nœud de prendre connaissance de la présence des puits secondaires dans le réseau et de transmettre les données collectées au puits le plus proche, nous proposons un mécanisme de routage basé sur la distance. Ce mécanisme de routage est une adaptation du protocole de routage proposé dans *DiscoProto* (section 2.5).

Dans la suite, nous supposons que chaque nœud du réseau a une adresse logique et est associé à un nœud père. Le mécanisme de routage multi-puits se déroule en deux étapes : l'étape de construction des chemins vers les puits secondaires et l'étape de routage et de maintenance des routes.

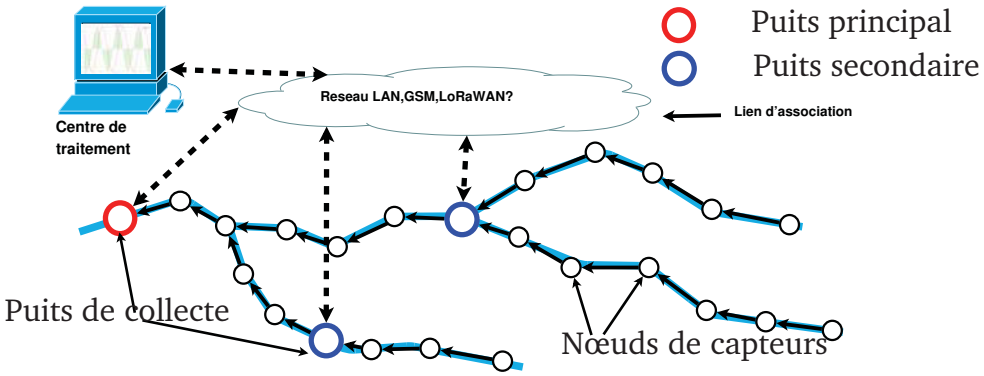



**Fig. 3.6.:** Réseau de capteurs sans fil avec plusieurs puits

### 3.4.1 Étape de construction des chemins

Lorsque des nœuds puits secondaires sont ajoutés au réseau et ont obtenu leur adresse logique, ils démarrent la diffusion *périodique* de messages d'annonce appelé *SADVERT* (Sink Advertisement) afin d'annoncer leur présence aux autres nœuds. Le message *SADVERT* contient l'adresse du nœud puits ($sinkAddr$) secondaire qui a émis le message, le nombre de sauts ($hopCount$) qui représente le nombre de nœuds traversés depuis le nœud puits ainsi que l'adresse du nœud ($nextHop$) à utiliser comme nœud relais pour atteindre le nœud puits. Afin d'éviter la diffusion des *SADVERT* à l'intégralité du réseau

qui risque d'engendrer une surcharge de trafic, un nœud puits ne peut pas retransmettre par un autre nœud puits.

Chaque nœud du réseau maintient une table *SADVERT* permettant de stocker le contenu des messages *SADVERT* reçus. Lorsqu'un nœud reçoit pour la première fois un message *SADVERT* d'un nœud puits, il crée une nouvelle entrée dans la table *SADVERT* contenant l'adresse du nœud puits( $sinkAddr$), le nœud relais ($nextHop$) qui sera utilisé pour l'acheminement des données collectées à ce puits et le nombre de sauts ($hopCount$) nécessaire pour cela. Dans le cas contraire, si la valeur $hopCount$ du message reçu est plus petite que celle contenue dans l'entrée de la table *SADVERT* alors le nœud met à jour les champs $netxtHop$ et $hopCount$ de l'entrée de la table. La figure 3.7 montre un exemple de propagation des messages *SADVERT* dans un réseau comportant trois puits.

Périodiquement, chaque nœud parcourt la table *SADVERT* et choisit comme puits de destination des données, celui correspondant à l'entrée ayant la plus petite valeur de $hopCount$. La valeur du $nextHop$ de cette entrée sera utilisée comme adresse de nœud relais pour atteindre le puits choisi.

**Algorithme 3.2:** Traitement à la réception d'un message SADVERT

**Result:** Réaction à la réception d'un message SADVERT.
**Input :** message SADVERT reçu d'un nœud relais

```
1  if isSink then
2      return ;
3  if TableSADVERT.trouver (SADVERT.sinkAddr) = ∅ then
       // réception du message pour la première fois
4      entreeSADVERT ← CreerEntree (SADVERT.sinkAddr
         SADVERT.sourceAddr,SADVERT.hopCount,TTL_MAX);
5      TableSADVERT.ajouter (entreeSADVERT);
6  else
7      entreeSADVERT ← TableSADVERT.trouver (SADVERT.sinkAddr);
8      if SADVERT.hopCount < entreeSADVERT.hopCount then
           // Meilleur chemin reçu
9          TableSADVERT.update (SADVERT.sinkAddr
             SADVERT.sourceAddr,SADVERT.hopCount,TTL_MAX);
10 SADVERT.hopCount ← SADVERT.hopCount + 1 ;
11 SADVERT.sourceAddr ← NodeAddr ;
12 Broadcast (SADVERT);
```

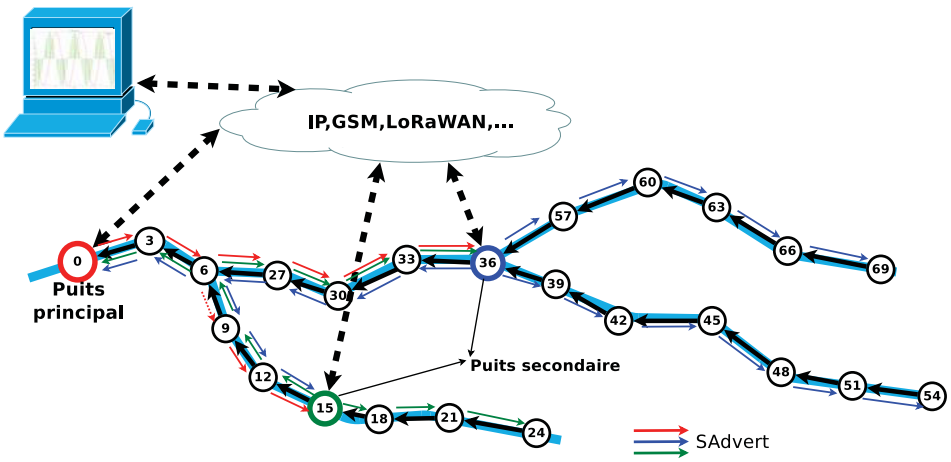



**Fig. 3.7.:** Propagation SADVERT

### 3.4.2 Maintenance des routes

Durant la durée de vie du réseau, des puits ou des nœuds relais peuvent cesser de fonctionner et donc cesser d'émettre des messages *SADVERT* vers d'autres nœuds du réseau. Afin d'assurer la cohérence de la table *SADVERT* et d'éviter qu'un puits non accessible soit choisi comme puits de destination des paquets de données, nous avons donné une durée de vie (*Age*) à chaque entrée de la table. Ainsi, une entrée qui n'a pas été mise à jour, jusqu'à atteindre l'age maximum, sera supprimée de la table *SADVERT* .

Pour ce faire, nous avons ajouté un champ $Age$ à chaque entrée de la table *SADVERT* représentant sa date (horodatage) de création ou de dernière mise à jour (voir table 3.6) en se basant sur l'horloge système du nœud. Lorsqu'un nœud reçoit un message *SADVERT*, il ajoute ou fait la mise à jour de l'entrée correspondante et réinitialise la valeur du champs $Age$.

| sinkAddr | nextHop | hopCount | Age |
|---|---|---|---|
| 0 | 3 | 2 | 90 |
| 15 | 9 | 3 | 57 |
| 36 | 6 | 4 | 75 |

**Tab. 3.6.:** Table SADVERT du nœud 6 de la figure 3.7

Périodiquement, chaque nœud vérifie la durée de l'entrée en comparant la différence de l'horloge système et de la date de l'entrée avec l'age maximum ($Age\_MAX$). Lorsqu'une entrée atteint la valeur $Age\_MAX$ définit

par défaut à $120\ s$, le nœud la supprime de la table *SADVERT*. Si la valeur du champs $sinkAddr$ correspond à l'adresse du puits utilisé par le nœud pour l'acheminement des données, alors le nœud choisit automatiquement les adresses du puits et du prochain nœud relais contenues dans l'entrée ayant le plus petit nombre de saut afin d'assurer la continuité du routage des données.

## 3.5 Évaluation du protocole *Dynamique DiscoProto*

Pour l'évaluation du protocole *Dynamic DiscoProto*, nous avons ajouté aléatoirement d'une part $50$, $100$, $150$, $200$ nœuds épars et d'autre part $5$, $10$, $15$, $20$, $25$ branches de $20$ nœuds à des réseaux existants de $50$, $100$, $150$, $200$, $250$ nœuds. Les différentes simulations effectuées sur l'environnement Omnet++/Castalia nous permettront d'étudier :

— Le temps d'association de l'ajout de nouveaux nœuds au réseau existant,

— L'influence de la taille d'un réseau existant sur le temps d'association de l'ajout de nouveaux nœuds,

— L'impact de la surcharge et de la distribution des messages de contrôle sur le réseau

— L'efficacité du protocole avec le taux d'association du réseau.

Pour ce faire, les nœuds épars se réveillent aléatoirement (selon la loi uniforme) après la phase de formation et d'adressage initiale du réseau. Cependant, les nœuds d'une même branche se réveillent tous au même moment.

Le tableau 3.7 récapitule les paramètres de simulation pour ces différentes évaluations.

### 3.5.1 Évaluation du coût de l'ajout de nouveaux nœuds

Pour cette évaluation, nous étudions la durée moyenne d'association des nouvelles branches de nœuds pour une taille de réseau fixe de $200$ nœuds.

| | |
|---|---|
| Temps de simulation | 2000, 4000, 6000, 8000, 7000 (s) |
| Nombre de scénarios | 1000 |
| Modèle de propagation | lognormal Shadowing ($\gamma = 2,4$ et $\sigma = 4,0$) |
| Puissance de transmission | 0 dBm |
| Sensitivité | -95 dBm |
| Couche Physique | 802.15.4 |
| Couche MAC | TMAC |

**Tab. 3.7.:** Paramètres de simulation

Tous les nœuds d'une même branche ajoutée aléatoirement après la formation du réseau sont démarrés au même moment.

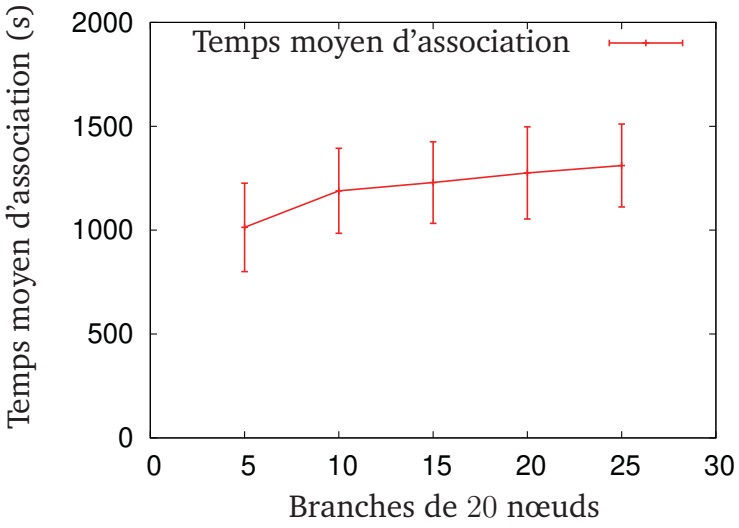



**Fig. 3.8.:** Temps d'association moyen par nombre de branches pour un réseau de 200 nœuds

La figure 3.8 montre que le nombre de nouvelles branches ajoutées au réseau n'a pas d'impact exponentiel sur le délai d'association pour une taille de réseau donnée. En effet, si les différentes branches intègrent le réseau grâce à des nœuds connectés qui ne sont pas soumis à l'influence de leurs challenges respectifs pour choisir le meilleur lien d'associations dans un même voisinage, alors dans ce cas les associations de ces branches se feront en parallèle. Ce qui a comme conséquence de réduire la durée moyenne d'association.

### 3.5.2 Influence de la taille du réseau initial sur l'ajout de nouveaux nœuds

Pour cette évaluation, nous allons étudier la durée moyenne d'association de 20 nouvelles branches de 20 nœuds à des réseaux de tailles différentes (100, 200, 300 et 400 nœuds). La figure 3.9 montre que le délai d'association ne dépend pas de la taille initiale du réseau. En effet, l'influence majeure de la taille du réseau initiale est dans la distance entre le point d'association et le Puits, qui n'a qu'une petite influence sur le temps total d'association.

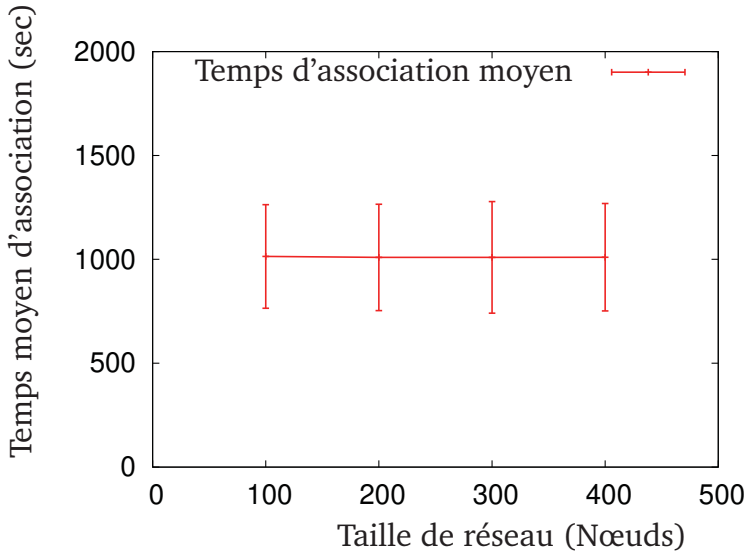



**Fig. 3.9.:** Temps moyen d'association d'ajouts de 20 branches de 20 nœuds sur des réseaux de tailles variées

### 3.5.3 Le taux d'association des nouveaux nœuds

Le tableau 3.8 montre que le taux d'association après l'ajout dynamique de branches de nœuds avoisine les 99.9%. Grâce à notre mécanisme d'ajout de nouveaux nœuds, *Dynamic DiscoProto* permet à ceux qui n'ont pas pu s'associer durant la phase initiale de formation du réseau de s'intégrer et de participer à la collecte de donnée. Cela permet de résoudre le problème de nœuds orphelins qui pourraient subsister avec *DiscoProto* et d'avoir un réseau pleinement connecté

| Nombre de branches ajoutées | 5 | 10 | 15 | 20 |
|---|---|---|---|---|
| Avant l'ajout de nouvelles branches | 99.747 | 99.630 | 99.687 | 99.688 |
| Après l'ajout de nouvelles branches | 99.955 | 99.896 | 99.884 | 99.860 |

**Tab. 3.8.:** Taux d'association moyen vs nombre de branches ajoutées

### 3.5.4 Étude de la surcharge et de la distribution des messages de contrôle

Durant la phase d'ajout de nouveaux nœuds au réseau existant, nous avons aussi évalué l'impact de la charge due aux messages de contrôle sur le réseau. La figure 3.10 montre que, lors de l'ajout de branches sur un réseau de 200 nœuds, le nombre de messages de contrôle croît linéairement en fonction du nombre de branches de 20 nœuds et donc du nombre de nouveaux nœuds ajoutés. Ceci est dû, en partie, au fait que chaque nouveau nœud répond par un *SonOffer* à chaque fois qu'il reçoit un *Fatheroffer* jusqu'à ce qu'il soit associé à son tour. Le nombre de messages tels que les *FatherOffer* et *ChallengeOffer* croît plus lentement que celui des *SonOffer* car ces messages sont diffusés uniquement par les nœuds déjà associés. Le nombre de nœuds associés augmente lui mécaniquement à chaque tour .

Cependant, les topologies avec beaucoup de branchements peuvent accroître le nombre de messages de contrôle tels que les messages *ChallengeOffer*. En effet, au niveau des zones de branchements où nous avons de très fortes concentrations de nœuds ou lorsqu'un nombre de nœuds épars est ajouté dans une même zone, chaque message *FatherOffer* diffusé entraîne la génération de beaucoup de messages *SonOffer* et chaque message *ChallengeOffer* diffusé peut être relayé par un grand nombre de nœuds connectés jusqu'à trois sauts. Ceci explique la proportion croissante du nombre de *SonOffer* par rapport au nombre de *ChallengeOffer* quand le nombre de branches augmente.

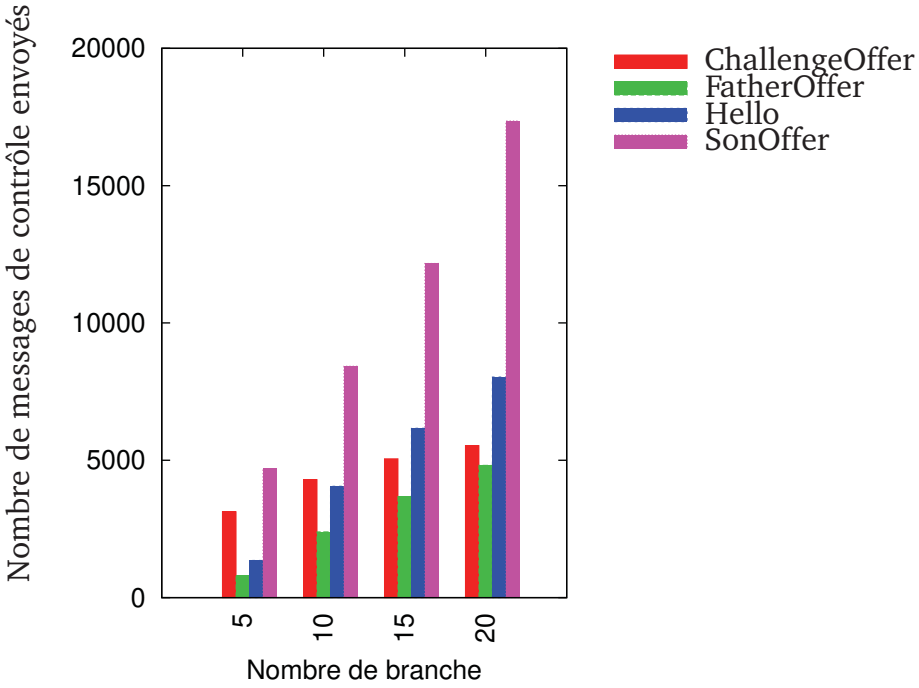



**Fig. 3.10.:** Distribution des messages de contrôle envoyés dans un réseau de 200 nœuds lors de l'ajout de branches

### 3.5.5 Évaluation du ratio et de la latence des paquets de données délivrés aux puits

Grâce au mécanisme *Dynamic DiscoProto*, il est possible, en fonction des besoins du réseau, d'ajouter des nœuds agissant comme des puits secondaires. Dans cette partie, nous évaluons l'impact de l'ajout de plusieurs puits dans le réseau sur le ratio et la latence des paquets de données délivrés aux puits. Pour ce faire, nous utilisons 1000 réseaux de 100 nœuds comportant 1, 5, 10, 15 et 20 puits. Après avoir obtenu leur adresse logique et choisi le puits à partir de leur table `SADVERT`, chaque nœud commence à générer du trafic durant 1000 secondes à raison de 10 paquets par minute. Durant nos simulations, les tailles maximum des files d'attente de la couche MAC sont définies à 32.

Les différentes évaluations montrent que le déploiement de plusieurs puits (principal et secondaires) dans les RdCSF linéaires permet d'améliorer le ratio et la latence de paquets de données délivrés aux puits. Sur la figure 3.11 avec un seul puits disponible sur le réseau, on remarque que plus de 50 % des données reçues par le puits arrive au delà des 600 *secondes* alors qu'avec le déploiement de plusieurs puis plus 75 % des données reçues par les puits

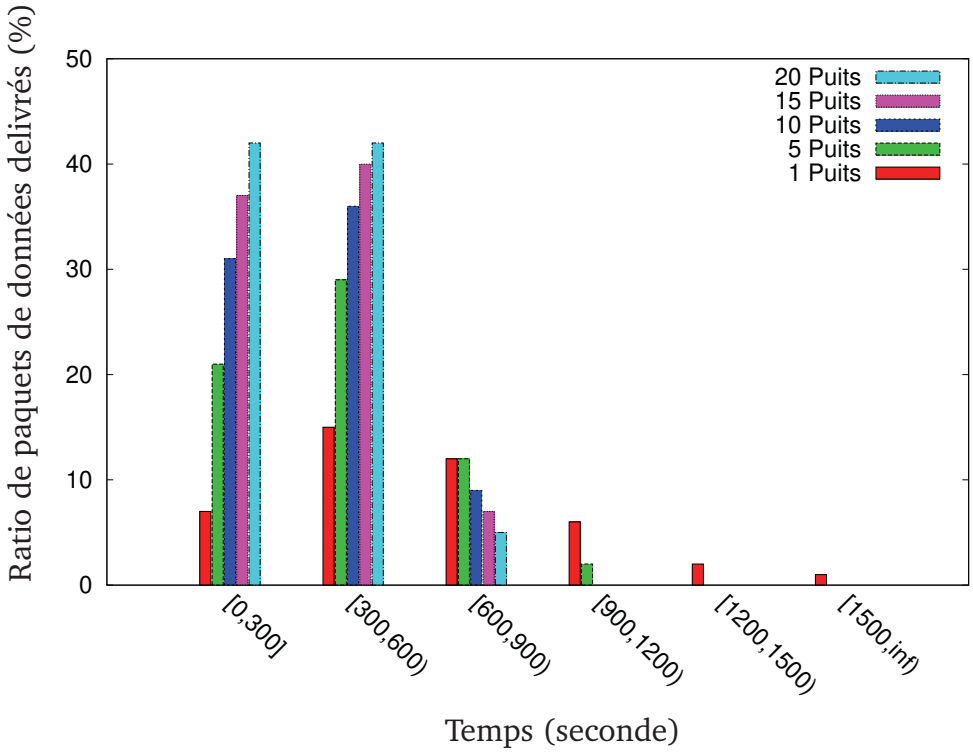



**Fig. 3.11.:** Latence des paquets de données délivrés pour un réseau de 100 nœuds

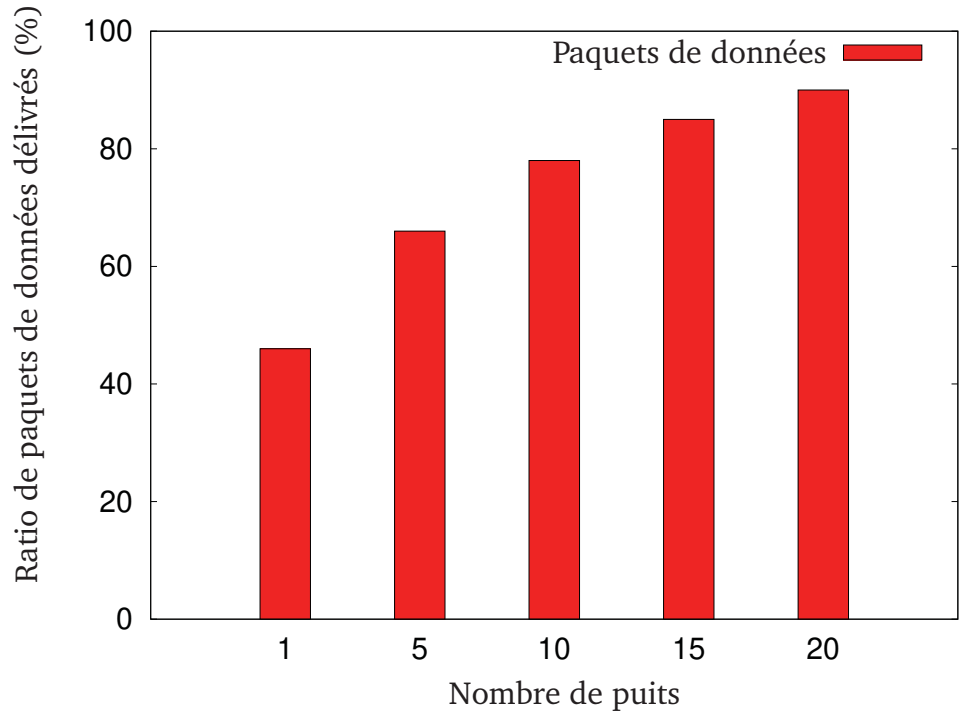



**Fig. 3.12.:** Ratio des paquets de données délivrés pour un réseau de 100 nœuds

arrive entre 0 et 600 secondes. En effet, le déploiement de plusieurs puits permet de partitionner le réseau et ainsi de diminuer le nombre de sauts pour atteindre la destination et réduisant ainsi la latence (voir la figure 3.11). La diminution du nombre de sauts a comme autre impact d'améliorer le ratio de paquets délivrés comme le montre la figure 3.12. En effet, le routage des données dans les RdCSF linéaires engendre une augmentation de trafic à chaque saut en direction du puits de destination (voir chapitre 3 à la page 122). Ceci occasionne des encombrements, des débordements de files d'attente et des collisions au niveau des nœuds au voisinage du puits. Réduire le nombre de sauts grâce au déploiement de plusieurs puits dans le réseau permet de réduire ces problèmes et donc d'améliorer le ratio comme le montre la figure 3.12.

### 3.5.6 Impact de la surcharge sur le réseau

Dans cette partie, nous évaluons la capacité d'un tel réseau à supporter la charge offerte. Pour ce faire, nous utilisons des réseaux de 100 nœuds comportant 1, 5, 10, 15 et 20 puits secondaires. Chaque nœud génère du trafic durant 1 000 secondes à différents débits de 10, 20, 40, 60 paquets par minute. La figure 3.13 montre que le ratio de paquets de données délivrés aux puits du réseau diminue très rapidement lorsque le ratio nombre de nœuds par puits dans le réseau est très grand et que le débit de transmission des nœuds augmente. En effet, la surcharge générée par l'augmentation du débit des données augmente les risques de collisions et de débordements de files d'attente au niveau des nœuds intermédiaires sur le chemin du puits. De plus, comme un nœud joue à la fois le rôle de capteur de données et de nœud relais, la quantité de données à transmettre est la somme du débit des données locales et des données reçues des autres nœuds l'utilisant comme relais. Ceci fait qu'au fur et à mesure qu'on se rapproche du puits, le débit des nœuds intermédiaires augmente très rapidement (Voir équation (3.1) à la page 122). Ces effets peuvent être atténués par une augmentation du nombre de puits disséminés dans le réseau qui permet de réduire les distances (nombre de sauts). Ceci permet alors de réduire le débit des nœuds relais en direction du puits choisi, les risques de débordement de files d'attente et du coup ceci permet d'améliorer le ratio de paquets de données délivrés aux puits.

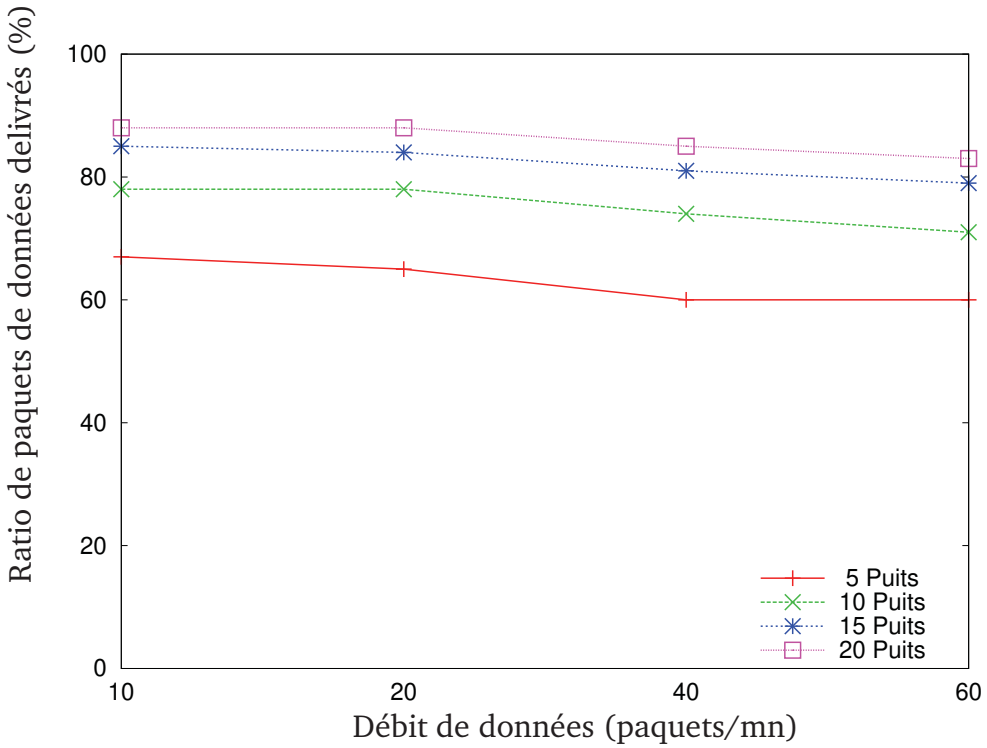



**Fig. 3.13.:** Ratio des paquets de données délivrés pour un réseau de 100 nœuds selon la charge offerte

Cependant, le déploiement de plusieurs puits, tout en réduisant le nombre de nœuds traversés par les paquets de données à destination du puits, ne permet pas d'éviter totalement les pertes de paquets causés par les collisions. Ce qui fait que le ratio de paquets de données n'atteint pas les $100\%$ même avec un ratio nombre de nœuds par puits égal à cinq. Ce type de problème est fortement lié à la couche MAC (TMAC dans notre cas) et ne peut être réglé par les couches supérieures. L'utilisation d'une couche MAC adaptée aux RdCSF linaires [SD15 ; Ndo+14b] permettrait peut-être de résorber ce problème.

## 3.6 Conclusion

Dans cette partie, nous avons présenté notre protocole de gestion dynamique d'un RdCSF Linéaire *Dynamic DiscoProto*. Il permet de gérer l'ajout de nouveaux nœuds épars et de nouvelles branches de nœuds. Il fournit également un mécanisme de gestion des réallocations d'adresses adapté à la linéarité du réseau. L'algorithme de routage implémenté supporte la réallocation de nouveaux blocs d'adresses et assure la continuité des services de routage des

données capturées aux différents puits du RdCSF linéaire. Les évaluations de l'ajout de nouveaux nœuds sur l'environnement de simulation Castalia/Omnet++ montrent que *Dynamic DiscoProto* s'adapte à de grands RdCSF linéaires et à l'ajout d'un grand nombre de nœuds épars ou de branches de nœuds mais peut occasionner des latences pendant l'intégration des nouveaux nœuds dans le cas de RdCSF linéaires denses c'est en particulier le cas pour des RdCSF linéaires qui possèdent beaucoup de nœuds situés dans des zones de branchements.

Grâce à son mécanisme d'ajout de nouveaux nœuds, *Dynamic DiscoProto* permet aussi de déployer des nœuds jouant le rôle de puits secondaire à l'initialisation ou après la formation du réseau afin de réduire le problème de congestion, de perte de paquets et de débordement de files d'attente au niveau des nœuds proches du puits dans le cas d'un RdCSF avec un seul puits. L'approche multi-puits permet d'améliorer le ratio de paquets de données délivrés aux différents puits. De plus, le déploiement de nœuds secondaires permet aussi de réduire le problème de latence dans l'acheminement des paquets de données observé dans un RdCSF linéaire avec un seul puits. Ces puits secondaires pourront utiliser d'autres technologies de transmission (GSM, IP, LoraWan,...) afin d'acheminer les données collectées vers le centre de collecte et de traitement.

Cependant, le déploiement de nœuds secondaires dans un RdCSF linéaire ne permet pas de résoudre tous les problèmes décelés lors de l'acheminement des données. Par exemple, pour réduire les congestions et les débordements de files d'attente, un mécanisme d'agrégation adapté aux RdCSF linéaires pourra être étudié. Pour réduire les pertes de paquets liés aux collisions, une intégration de *Dynamic DiscoProto* avec un protocole MAC adapté aux RdCSF linéaires comportant des branchements pourra être étudiée et évaluée.

# 4 Conclusion et Perspectives

Les RdCSF linéaires sont des réseaux de capteurs pour lesquels les nœuds sont déployés le long d'une ou de plusieurs lignes reliées par des branchements. Les RdCSF linéaires sont utilisés pour la surveillance de milieux tels que des cours d'eau, des canalisations des fluides (eau, gaz, pétrole,etc.), des chemins de fer, des autoroutes, etc. La topologie particulière des RdCSF linéaires ne permet pas aux différentes mécanismes d'adressage décrits dans l'état de l'art de former la topologie et de déployer des adresses de façon permanent, ceci automatiquement dans un RdCSF linéaire quelconque

Dans la suite, nous résumons les différentes contributions de notre thèse permettant la formation et l'adressage de RdCSF linéaires et présentons des perspectives qui pourraient consolider et étendre nos contributions.

## 4.1 Résumé des contributions

Dans cette thèse, nous avons fait l'étude des RdCSF linéaires et proposé une nouvelle définition en introduisant une métrique appelée *facteur de linéarité* (*lF*) qui représente la valeur seuil du pourcentage des nœuds simples dans un RdCSF linéaire c'est à dire des nœuds ayant au plus $2K$ voisins, $K$ représentant le coefficient de redondance du RdCSF linéaire. Un RdCSF sera considéré comme étant un RdCSF linéaire si le pourcentage de nœuds simples est supérieur ou égal au facteur de linéarité (*lF*).

Durant nos travaux, deux contributions ont été proposées. La première concerne un mécanisme de découverte de voisinage des nœuds, de formation initiale d'une topologie, d'adressage et de routage d'un RdCSF linéaire appelé *DiscoProto*. *DiscoProto* assure la formation logique d'un RdCSF linéaire sans connaissance au préalable des caractéristiques du réseau telles que la profondeur, le nombre maximum de nœuds fils par nœud ou la position des nœuds. Grâce à la seule connaissance du voisinage, *DiscoProto* permet de

découvrir et de former une topologie logique en arbre avec le minimum de branchements et sans limitation de profondeur, mais aussi de distribuer les adresses de façon homogène dans le réseau. Pour ce faire, *DiscoProto* se base sur la valeur d'un coût (*Objectif*) local associé à chaque lien entre deux nœuds qui est un indicateur de proximité afin de construire un arbre linéaire recouvrant de coût maximum dont la racine est le puits du RdCSF linéaire. Même s'il est de type algorithme Glouton, *DiscoProto* a la capacité d'effectuer la construction de l'arbre parallèlement sur les différents branches de nœuds du RdCSF linéaire afin d'améliorer le temps de formation de la topologie. Les résultats des évaluations réalisées sur l'environnement de simulation Omnet++/Castalia ont montré que les topologies logiques formées par *DiscoProto* sont proches de la topologie physique du RdCSF linéaire.

Après la phase de formation logique, *DiscoProto* fournit un *adressage initial* hiérarchique des nœuds basé sur la taille réelle du réseau obtenue grâce à la remontée des tailles des sous-arbres jusqu'au puits. Ce mécanisme d'adressage fournit, à chaque nœud de capteurs du réseau, une adresse unique ainsi qu'un bloc d'adresse de réserve $Fskip$ pour l'ajout éventuel d'autres nœuds. Cette technique d'adressage a permis de proposer un mécanisme de routage basé sur la connaissance de l'adresse de destination sans utilisation de messages de contrôle pour la découverte de routes vers la destination. Ces résultats ont aussi montré que *DiscoProto* s'adapte aux RdCSF linéaires ayant un grand nombre de branchements. Cependant, une latence dans la phase de formation du réseau est observée dans le cas de RdCSF linéaires comportant aucun ou peu de branchements par rapport aux RdCSF linéaires ayant beaucoup de branchements.

Les résultats des évaluations de la phase d'adressage montrent que *DiscoProto* assure un adressage effectif en fonction de la taille réelle du réseau et une bonne répartition des adresses grâce à la connaissance de la taille de chaque sous-arbre de nœuds tout en évitant le plus souvent l'apparition de nœuds orphelins. En effet, *DiscoProto* profite de la phase de propagation des adresses initiée par le puits à destination de l'ensemble des nœuds pour renseigner et maintenir la table de routage. Cette approche permet ainsi d'assurer l'acheminement des paquets échangés entre les nœuds du réseau et le puits sur la seule connaissance de l'adresse de destination.

Notre deuxième contribution porte sur un mécanisme appelé *Dynamic DiscoProto* permettant l'extension d'un RdCSF linéaire existant grâce à l'ajout d'un nombre quelconque de nouveaux nœuds épars ou de branches de nœuds. Pour ce faire, *Dynamic DiscoProto* s'appuie sur les fonctionnalités de découverte et de formation de topologie d'un RdCSF linéaire de *DiscoProto* et propose une mécanisme pour ajouter des nœuds épars ou des branches entières de nœuds. *Dynamic DiscoProto* fournit un mécanisme de réallocation d'adresses lorsqu'un nœud a épuisé son bloc de réserve d'adresses $Fskip$ après avoir ajouté de nouveaux nœuds afin de le réapprovisionner. Lorsque le bloc $Fskip$ est insuffisant pour adresser une nouvelle branche de nœuds, le mécanisme de réallocation permet au puits de calculer un nouveau bloc d'adresses en fonction du nombre de nœuds de la nouvelle branche et de la valeur de $Fskip$. À partir du nouveau bloc d'adresses reçu, *Dynamic DiscoProto* garantit un adressage hiérarchique à l'intérieur de chaque nouvelle branche de nœuds.

Les résultats de l'évaluation de l'ajout des nœuds montrent que *Dynamic DiscoProto* associe tous les nouveaux nœuds et arrive aussi à intégrer les nœuds orphelins qui sont apparus pendant la phase de formation initiale du réseau. Ces résultats ont aussi montré que *Dynamic DiscoProto* s'adapte aux grands RdCSF linéaires et ajoute, quelque soit leur nombre, les nœuds épars sans une augmentation exponentielle du temps d'association. Cependant une latence sur le temps d'association des nœuds est observée lorsque les nouvelles branches se chevauchent (provoquant des interférences mutuelles).

*Dynamic DiscoProto* permet aussi l'ajout de nœuds puits secondaires dans un RdCSF linéaire et fournit un mécanisme de routage basé sur le nombre de sauts. Il permet l'acheminement des données à destination de ces nœuds puits. L'ajout des nœuds puits secondaires est une façon de réduire les problèmes de congestion et de latence rencontrés généralement dans les RdCSF linéaires. L'évaluation du routage des données à destination des puits d'un RdCSF linéaire montre que notre mécanisme améliore le ratio de données délivrées aux puits tout en diminuant la latence. Cependant notre approche de déploiement de plusieurs puits ne permet pas de résoudre tous les problèmes rencontrés dans les RdCSF linéaires en particulier les problèmes de collisions liés à la couche MAC utilisée qui ne tire pas partie des spécificités de ces topologies.

## 4.2 Perspectives

### Validation par maquettage

Pour la suite de nos travaux de recherche, nous envisageons de déployer nos protocoles sur des nœuds réels. Cette phase expérimentale devrait permettre d'ajuster plus finement les paramètres définis dans les mécanismes que nous avons proposés (les valeurs des différents temps par exemple). Ce couplage "simulation-maquettage" devrait nous aider à améliorer le fonctionnement de nos protocoles *DiscoProto* et *Dynamic DiscoProto*.

### Déploiement en environnement réel

Après cette étape d'ajustement, notre projet majeur est l'implémentation de nos différentes contributions sur des nœuds de capteurs afin de les déployer en environnement réel en collaboration avec des équipes de recherche des Universités du Sénégal. Pour ce faire, nous prévoyons dans un premier temps d'implémenter et d'évaluer nos différents protocoles sur des nœuds de capteurs. Dans un deuxième temps, nous déploierons ces nœuds de capteurs pour la surveillance environnementale du fleuve du Sénégal dans la région de St-Louis qui nous semble être un site de choix pour la mise en œuvre d'un RdCSF linéaire. En effet, en plus d'être la principale source d'eau potable avec le lac de Guiers, la région du fleuve du Sénégal à Saint-Louis (sur environ $100\ km$) est une zone de fortes activités agricoles. La mesure des données telles que la qualité de l'eau (détection de pollution chimiques), débit des irrigations (gestion rationnelle des eaux), niveau du lit du fleuve (prévention risque d'inondation) permettra un meilleur suivi en temps réels de ces différentes activités. Pour ce type d'application, les données applicatives de taille 50 à 200 octets pourront être capturées à des fréquences $15$ à $30$ minutes et la moyenne calculée sera transmise toutes les heures au centre de collecte. Un système d'envoi d'alertes déclenchées permettra aussi d'envoyer des messages lorsqu'une capture avec un écart significatif est effectuée.

## Amélioration du ratio de paquets délivrés

Nous avons vu dans *Dynamic DiscoProto* que malgré le déploiement de puits secondaires, des pertes de paquets de données subsistent du fait des collisions engendrées par le mécanisme d'accès au canal radio. Nous proposons d'étudier l'adaptation d'un protocole MAC spécialement dédié aux RdCSF linéaires tel que [Ndo+16 ; Fan+11] avec nos différentes contributions afin d'évaluer leurs impacts sur le ratio de paquets de données délivrés au puits.

Une autre perspective d'amélioration du ratio de paquets délivrés est d'étudier et d'implémenter une solution d'agrégation des données adaptée aux RdCSF linéaires afin de réduire les pertes de paquets liés aux débordements de files d'attente des nœuds relais au voisinage du puits. Ce phénomène est très fréquent dans les RdCSF linéaires car la quantité de données à acheminer à un puits augmente rapidement au fur et à mesure que l'on se rapproche de lui (voir section 3.1 à la page 122). Agréger les données permettra de réduire les transmissions des nœuds du chemin linéaire à destination du puits ce qui devrait avoir comme conséquence la réduction des congestions et des cas de débordements de files d'attente, ceci devrait conduire à l'amélioration des ratios de paquets délivrés. Une agrégation pertinente des données est plutôt dépendant du contexte applicatif et de la nature des informations à véhiculer. Pour de la surveillance environnementale, la redondance des données produites laisse espérer des gains en performances élevés.

## Intégration des technologies LPWAN

Nous envisageons dans nos travaux futurs, d'étudier la faisabilité de l'intégration de nos différentes contributions avec la technologie *LoRa* et une couche MAC adaptée aux RdCSF linéaires. Cette intégration permettra de profiter des avantages de la portée de transmission permise par *LoRa* et ainsi d'avoir des distances beaucoup plus grandes entre les nœuds du réseau pour couvrir de très grandes zones à surveiller. Nous présentons alors des problèmes au niveau de la quantité de données à transmettre.

# A

# Générateur de topologies linéaires

```
1
2
3  #include "LinearNoMobilityManager.h"
4  #include <vector>
5  #include <cassert>
6  #include <cmath>
7
8
9  Define_Module(LinearNoMobilityManager);
10
11 std::vector<LinearNoMobilityManager::Turtle*> LinearNoMobilityManager::
       turtles;
12 Position LinearNoMobilityManager::dimension;
13 Position LinearNoMobilityManager::move;
14
15 double LinearNoMobilityManager::deltaLength;
16 double LinearNoMobilityManager::deltaRotation;
17 double LinearNoMobilityManager::forkProbability;
18 double LinearNoMobilityManager::offProbability;
19 double LinearNoMobilityManager::branchOffProbability;
20 int LinearNoMobilityManager::minSleepDuration;
21 int LinearNoMobilityManager::maxSleepDuration;
22 int LinearNoMobilityManager::offBranch;
23 int LinearNoMobilityManager::offNode;
24
25
26 /// Initialize thr module
27 void LinearNoMobilityManager::initialize()
28 {
29         node = getParentModule();
30         index = node->getIndex();
31         network = node->getParentModule();
32         wchannel = network->getSubmodule("wirelessChannel");
33
34         if (!network)
35                 opp_error("Unable to obtain SN pointer for deployment
                      parameter");
36
37         if (!wchannel)
```

```
38                opp_error("Unable to obtain wchannel pointer");
39
40        string deployment = network->par("deployment");
41        if ( deployment == "linear" ){
42                parseLinearDeployment();
43        }else{
44                parseDeployment();
45        }
46 }
47
48 void LinearNoMobilityManager::parseLinearDeployment() {
49     // Pan Coordinator
50
51     if ( index == 0 ) {
52         dimension.x = network->par("field_x");
53         dimension.y = network->par("field_y");
54         dimension.z = network->par("field_z");
55
56         // Random variation in angle and length for each step
57         deltaRotation = par("deltaRotation");
58         deltaLength = par("deltaLength");
59
60         // Probability for a line to fork (create a new branch)
61         forkProbability = par("forkProbability");
62
63         // Probability that a new node is off (ie., will be switched on
               later)
64         offProbability = par("offProbability");
65
66          // Probability that a new branch is off (ie., will be switched on
               later)
67         branchOffProbability = par("branchOffProbability");
68
69         minSleepDuration = par("minSleepDuration");
70         maxSleepDuration = par("maxSleepDuration");
71
72         // Initial step size
73         move.x = par("distance");
74         move.y = 0;
75         move.z = 0;
76
77         //Count offNode and offBranch
78         offNode = 0;
79         offBranch = 0;
80
```

```
81          Position center(dimension);
82          center/=2;
83          turtles.push_back(new Turtle(dimension, center, move ) );
84      }
85
86      fromFork = false;
87
88          Turtle *turtle = chooseTurtle() ;
89
90      bool isAloneOff = uniform(0,100) < offProbability ;
91      if ( turtle->getOffNodes() || isAloneOff ) {
92        m_isPowered = false;
93        if ( turtle->getOffNodes() )
94          sleepDuration = turtle->getSleepDuration();
95        if ( isAloneOff )
96          sleepDuration = uniform ( minSleepDuration, maxSleepDuration );
97
98        offNode++;
99      }
100     else {
101       m_isPowered = true;
102         }
103     // Coordinator is ALWAYS on
104     if ( index == 0 )
105       m_isPowered = true;
106
107     Position newPos;
108
109     if ( isAloneOff )
110       newPos = turtle->stepSide( );
111     else
112       newPos = turtle->step( true );
113
114         nodeLocation.x = newPos.x;
115         nodeLocation.y = newPos.y;
116         nodeLocation.z = newPos.z;
117
118         return ;
119 }
```

# B

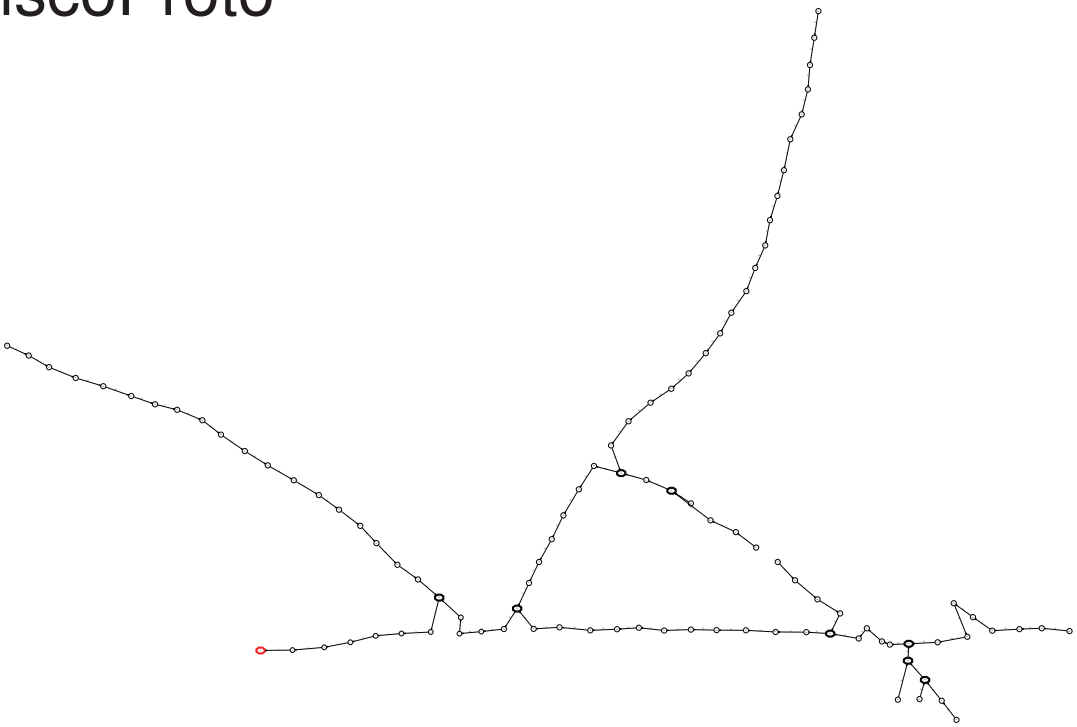
# Exemple de topologies générées utilisées pour nos simulations sur Castalia

Les exemples de la figure B.1 représentent des topologies de RdCSFL générées par notre générateur de topologie aléatoire et qui seront utilisés dans notre simulateur pour la validation de notre protocole DiscoProto.

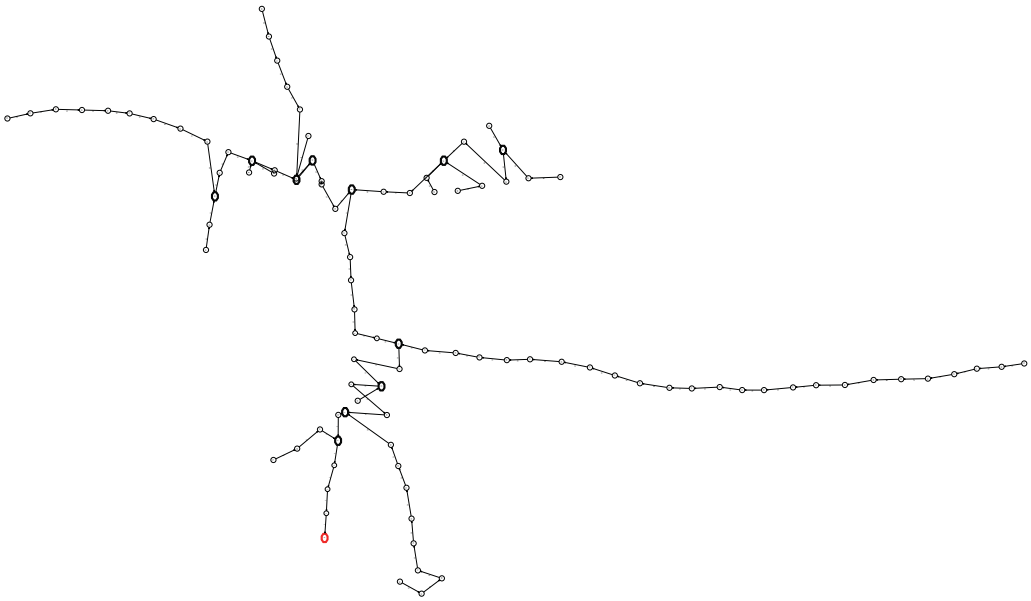
Fig. B.1.: Exemple de topologies générées de 100 nœuds

# C

# Exemple de topologies formées grâce à DiscoProto



(a)



(b)

**Fig. C.1.:** Exemples de topologies formées par DiscoProto

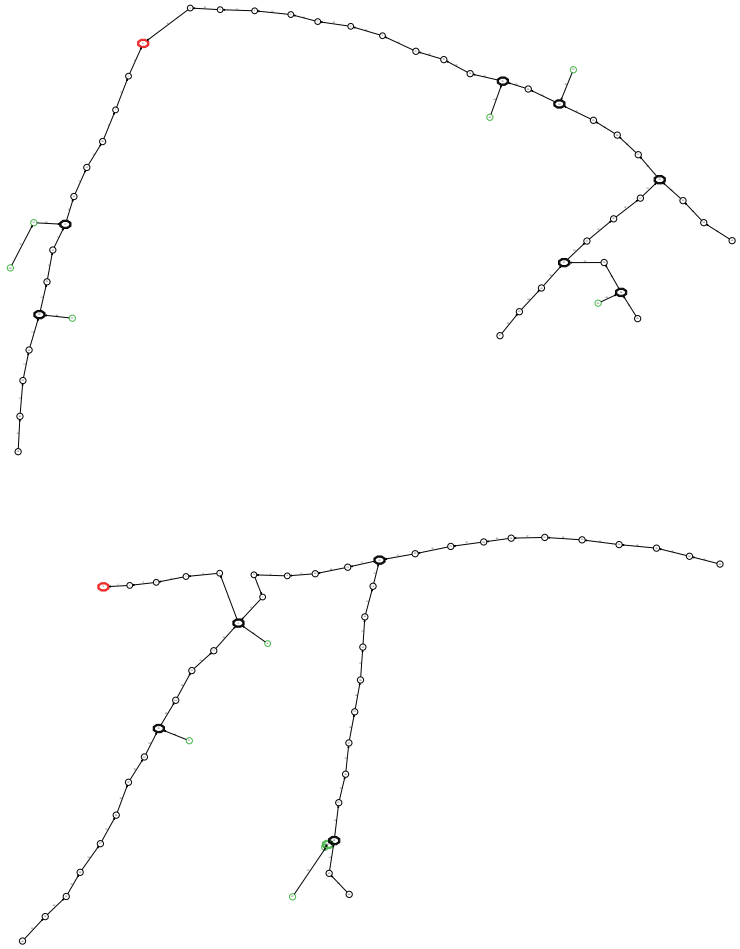


**Fig. C.2.:** Exemples de topologies formés par Dynamic DiscoProto aprés l'ajout de nouveaux épars(nœuds verts)

**Fig. C.3.:** Exemples de topologies formés par Dynamic DiscoProto avec l'ajout de branches de nœuds (nœuds verts)

# D

# Vérificateur de la linéarité des topologies

```
1  class LinearityChecker
2    NODEREG=%r{ *<text x = "([0-9]+)" y = "([0-9]+)" .*> *([0-9]+)
         *</text> *}
3
4    def initialize(args)
5      @graphs=[]
6      @mode=:list
7      @options = {}
8      @options[:min]=0
9      @options[:max]=100
10     @options[:range]=84
11     @options[:neighbourgs]=6
12     @options[:sorted]=false
13     @options[:verbosedisplay]=false
14     @options[:graphical]=nil
15
16     opts=OptionParser.new do |opts|
17       opts.banner = "Usage: linearitycheck.rb [options]
           fichiers_svg"
18       opts.on("-l", "--list", "list mode (default): List all svg
            files with the linearity computed") { @mode=:list }
19       opts.on("-g", "--graphical TOOL", String, "graphical
           display with given tool") { |tool| @options[:graphical
           ]=tool }
20       opts.on("-v", "--verbose", "verbose: display linearity in
           addition to name") { @options[:verbosedisplay]=true }
21       opts.on("-f", "--filter", "filter mode : select graphs by
           linearity score (see -m and -L)") { @mode=:filter }
22       opts.on("-s", "--sorted", "Sort the results by linearity")
            { @options[:sorted]=true }
23       opts.on("-n", "--neighbourgs NEIGHBOURGS", Integer, "
           Maximum number of neighbhours allowed [default #{
```

```
        @options[:neighbourgs]}]") { |neighbourgs| @options[:
        neighbourgs]=neighbourgs }
24      opts.on("-m", "--min MIN", Numeric, "Minimum percentage of
         nodes [default #{@options[:min]}] with too many
        neighbourgs (implies filter mode)") { |min| @options[:
        min]=min; @mode=:filter; }
25      opts.on("-M", "--max MAX", Numeric, "Maximum percentage of
         nodes [default #{@options[:max]}] with too many
        neighbourgs (implies filter mode)") { |max| @options[:
        max]=max; @mode=:filter; }
26      opts.on("-r", "--range RANGE", Numeric, "Average range in
        meters [default #{@options[:range]}] ") { |range|
        @options[:range]=range }
27      opts.on("-h", "--help", "Prints this help") { $stderr.puts
        (opts); exit(1); }
28    end
29
30    opts.parse!(args)
31    @files=args
32  end
33
34  def process
35    check_linearity_all
36
37    case @mode
38              when :list
39                @options[:verbosedisplay]=true
40                selected=@graphs
41              when :filter
42                selected=@graphs.select { |g|
43                            g.linearity_score >= @options[:min]
                                 && g.linearity_score <=
                               @options[:max]
44                }
45    end
46
47    sort_by_linearity(selected) if @options[:sorted]
48
```

```
49      if @options[:verbosedisplay] then
50        displayer=Proc.new{ |g| "#{g.name}: #{g.linearity_score}"}
51      else
52        displayer=Proc.new{ |g| g.name }
53      end
54
55      selected.each do |g|
56        puts displayer.call(g)
57        if @options[:graphical] then
58          `#{@options[:graphical]} #{g.name}`
59        end
60      end
61
62    end
63
64    def check_linearity(svgfile)
65      graph = Graph.new(svgfile)
66      File.open(svgfile) { |f|
67        f.each_line{ |line|
68          if ( line =~ NODEREG ) then
69            graph.add(Node.new($3,Position.new($1.to_f/1.9/1.9,$2.
                  to_f/1.9)))
70          end
71        }
72      }
73      graph.linearity(@options[:range], @options[:neighbourgs])
74      @graphs << graph
75      return graph
76    end
77
78    def check_linearity_all
79      @files.each { |arg| check_linearity(arg) }
80    end
81
82    def sort_by_linearity(selected)
83      selected.sort! { |g1, g2| g1.linearity_score <=> g2.
          linearity_score }
84    end
```

```
85
86    def vizualize
87      @graphs.each { |g|
88        puts g
89        `ristretto #{g.name}`
90      }
91    end
92  end
93
94  lc=LinearityChecker.new(ARGV)
95  lc.process
```

# E

# Exemple de paramètres de simulation sur Castalia pour l'évaluation de DiscoProto

```
# ************************************************************
# *  Fichier de simulation DiscoProto *
# * *
# ************************************************************

[Config Seed]
seed-0-mt = 1025

[Config Simtime]
sim-time-limit = 3600s # 50 secs of data + 1 sec of MAC setup

[Config Debit]
SN.node[*].Application.packet_rate = 0.0167 #(1 packet tous les
    10 mn)

[General]

# ============================================================
# Always include the main Castalia.ini file
# ============================================================
include ../Parameters/Castalia.ini

SN.field_x = 4000 # meters
SN.field_y = 4000 # meters
SN.numNodes = 100
SN.deployment = "linear"
SN.node[*].MobilityManagerName = "LinearNoMobilityManager"
SN.node[*].MobilityManager.distance = 20
```

```
28 SN.node[*].MobilityManager.deltaLength = 0.2
29 SN.node[*].MobilityManager.forkProbability = 10.0
30 SN.node[*].MobilityManager.offProbability = 0.0
31 SN.node[*].MobilityManager.branchOffProbability = 0.0
32 SN.node[*].MobilityManager.minSleepDuration = 1000
33 SN.node[*].MobilityManager.maxSleepDuration = 1100
34
35 # Select a Radio and a default Tx power
36 SN.node[*].Communication.Radio.RadioParametersFile = "../
      Parameters/Radio/CC2420.txt"
37 SN.node[*].Communication.Radio.TxOutputPower = "-3dBm"
38 SN.node[*].Communication.Radio.CCAthreshold = -92.0
39 # lognormal shadowing model parameters
40 SN.wirelessChannel.pathLossExponent = 2.4
41 SN.wirelessChannel.sigma = 4.0
42
43 #Application parameters
44 SN.node[*].ApplicationName = "ThroughputTest"
45 SN.node[*].Application.dataDuration = 1000 #(en sec)
46 SN.node[*].Application.latencyHistogramMax = 3000
47 SN.node[0].Application.packetHeaderOverhead = 5 // in bytes
48 SN.node[0].Application.constantDataPayload = 35 // in bytes
49
50 [Config TopoDisco]
51 SN.node[*].Communication.RoutingProtocolName = "
      DiscoProtoRouting"
52 SN.node[0].Communication.Routing.isNetCoord = true # PAN
      Coordinator
53 SN.node[0].Communication.Routing.isCSink = true # PAN
      Coordinator
54 SN.node[*].Communication.Routing.helloCycleDelay = 4000 #in msec
       delay bitween two sent hellos
55 SN.node[*].Communication.Routing.collectSonDelay = 30000 #in
      msec Time before start STATE Collection phase
56 SN.node[*].Communication.Routing.sonOfferDelay = 3000 #in msec
      time to collect son's nodes
57 SN.node[*].Communication.Routing.chalengeResponseDelay = 3000 #
      in msec
```

```
58 SN.node[*].Communication.Routing.sendFatherOfferDelay = 500 #in
       msec delay between two sent fatherOffers
59 SN.node[*].Communication.Routing.collectSonRetry = 2 # nbre of
       time to execute Collecting Son if no received sonOffer
60 SN.node[*].Communication.Routing.maxFatherOfferPacketRetry = 2
61 SN.node[*].Communication.Routing.availabilityCoefficient = 2 #
       Coefficient of Availability Fskip
62 SN.node[*].Communication.Routing.netBufferSize = 32
63
64 [Config TMAC]
65 SN.node[*].Communication.MACProtocolName = "TMAC"
66 SN.node[*].Communication.MAC.phyDataRate = 250
67 SN.node[*].Communication.MAC.macBufferSize = 32
68 SN.node[*].Communication.MAC.maxTxRetries = 4
69 SN.node[*].Communication.MAC.listenTimeout = 61
70 SN.node[*].Communication.MAC.disableTAextension = false
71 SN.node[*].Communication.MAC.conservativeTA = true
72
73
74 [Config CSMA]
75 include ../Parameters/MAC/CSMA.ini
```

# Liste des publications

## Journals

1. **Moussa Dethie Sarr**, François Delobel, Michel Misson, Ibrahima Niang. *Automatic and dynamic network establishment for linear WSNs*. Dans Wireless Networks Journal. Novembre 2017. DOI : 10.1007/s11276-017-1600-4.

## Conférences Internationales

1. **Moussa Dethie Sarr**, François Delobel, Michel Misson, Ibrahima Niang. *Robust discovery, addressing and routing protocol for dynamic linear network*. Dans International Symposium on Wireless Communication Systems (ISWCS). Août 2017. DOI : 10.1109/ISWCS.2017.8108099.
2. **Moussa Dethie Sarr**, François Delobel, Michel Misson, Ibrahima Niang. *Automatic discovery of topologies and addressing for Linear wireless sensors networks*. Dans IFIP Wireless Days (WD). Novembre 2012. DOI : 10.1109/WD.2012.6402801.

## Conférences Nationales

1. **Moussa Dethie Sarr**, François Delobel, Michel Misson, Ibrahima Niang. *Découverte automatique de topologies pour les réseaux de capteurs sans-fils linéaires - Implémentation*. Dans Journées Nationales des Communications Terrestres (JNCT). Juin 2015.

# Bibliographie


[AB+16] Ammar AMIRA BEN, Dziri ALI, Terre MICHEL et Youssef HABIB. „Multi-hop LEACH based cross-layer design for large scale wireless sensor networks“. In : *2016 International Wireless Communications and Mobile Computing Conference IWCMC)*. 2016, p. 763–768 (cf. p. 31, 34).

[Aky+02] Ian F. AKYILDIZ, Weilian SU, Yogesh SANKARASUBRAMANIAM et Erdal CAYIRCI. „A survey on sensor networks“. In : *IEEE Communications magazine* 40.8 (2002), p. 102–114 (cf. p. 1).

[All08] ZigBee ALLIANCE. *ZigBee Specification*. ZigBee Standard Organization. ZigBee, 2008 (cf. p. 4, 5, 14, 18, 48, 49).

[AM08] Massimo ARATTANO et Lorenzo MARCHI. „Systems and Sensors for Debris-flow Monitoring and Warning“. In : *Sensors* 8.4 (2008), p. 2436–2452 (cf. p. 11).

[AR15] Fayaz AKHTAR et Mubashir Husain REHMANI. „Energy replenishment using renewable and traditional energy resources for sustainable wireless sensor networks : A review“. In : *Renewable and Sustainable Energy Reviews* 45 (2015), p. 769–784 (cf. p. 2).

[Ath] Boulis ATHANASSIOS. *Castalia : A Simulator for Wireless Sensor Networks*. https ://castalia.forge.nicta.com.au (cf. p. 7, 106).

[Azi+13] A. A. AZIZ, Y. A. SEKERCIOGLU, P. FITZPATRICK et M. IVANOVICH. „A Survey on Distributed Topology Control Techniques for Extending the Lifetime of Battery Powered Wireless Sensor Networks“. In : *IEEE Communications Surveys & Tutorials* 15.1 (2013), p. 121–144 (cf. p. 26).

[Bak+07] C. R. BAKER, K. ARMIJO, S. BELKA et al. „Wireless sensor networks for home health care“. In : *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*. T. 2. 2007, 832–837 (cf. p. 1, 10).

[Bos+16] Adrien van den BOSSCHE, Eric CAMPO, Jenny DUCHIER et al. „Multi-dimensional Observation Methodology for the Elderly in an Ambient Digital Environment“. In : *Computers Helping People with Special Needs - 15th International Conference, ICCHP 2016, Linz, Austria, July 13-15, 2016, Proceedings, Part I*. 2016, p. 285–292 (cf. p. 10).

[Cas+08] Kenan CASEY, Alvin S. LIM et Gerry V. DOZIER. „A Sensor Network Architecture for Tsunami Detection and Response“. In : *IJDSN* 4.1 (2008), p. 28–43 (cf. p. 11).

[CL13] Yu-Fang CHUNG et Chia-Hui LIU. „Design of a Wireless Sensor Network Platform for Tele-Homecare“. In : 13 (2013), p. 17156–75 (cf. p. 10).

[Cun07] André CUNHA. *On the use of IEEE 802.15.4/ZigBee as federating communication protocols for wireless sensor networks*. Rapp. tech. Polytechnic Institute of porto (ISEP-IPP), 2007 (cf. p. 18, 49).

[DL03] Tijs van DAM et Koen LANGENDOEN. „An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks“. In : *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*. SenSys '03. ACM, 2003, p. 171–180 (cf. p. 107).

[Fan+11] Chen FANG, Hao LIU et LiLi QIAN. „LC-MAC : An Efficient MAC Protocol for the Long-Chain Wireless Sensor Networks“. In : IEEE, 2011, p. 495–500 (cf. p. 45, 66, 151).

[Far+10] M. O. FAROOQ, A. B. DOGAR et G. A. SHAH. „MR-LEACH : Multi-hop Routing with Low Energy Adaptive Clustering Hierarchy“. In : *2010 Fourth International Conference on Sensor Technologies and Applications*. 2010, p. 262–268 (cf. p. 31, 33).

[FE17] S. FARREL et ED. *Low Power Wide Area Network Overview*. IETF. IETF, 2017 (cf. p. 13, 20).

[Fra+09] Michael FRASER, Ahmed ELGAMAL, Xianfei HE et Joel P. CONTE. „Sensor network for structural health monitoring of a highway bridge“. In : *Journal of Computing in Civil Engineering* 24.1 (2009) (cf. p. 12).

[Gao+08] Tia GAO, Christopher PESTO, Leo SELAVO et al. „Wireless medical sensor networks in emergency response : Implementation and pilot results“. In : *Technologies for Homeland Security, 2008 IEEE Conference on*. IEEE, 2008, p. 187–192 (cf. p. 10).

[GB08] James M. GILBERT et Farooq BALOUCHI. „Comparison of energy harvesting systems for wireless sensor networks“. In : *International Journal of Automation and Computing* 5.4 (2008) (cf. p. 11).

[GP10] Carles GOMEZ et Josep PARADELLS. „Wireless home automation networks : A survey of architectures and technologies“. In : *IEEE Communications Magazine* 48.6 (2010) (cf. p. 12).

[Gut+09] Álvaro GUTIÉRREZ, Carlos GONZÁLEZ, Javier JIMÉNEZ-LEUBE et al. „A Heterogeneous Wireless Identification Network for the Localization of Animals Based on Stochastic Movements“. In : *Sensors* 9.5 (2009), p. 3942–3957 (cf. p. 11).

[Hei+00] Wendi Rabiner HEINZELMAN, Anantha CHANDRAKASAN et Hari BALAKRISHNAN. „Energy-Efficient Communication Protocol for Wireless Microsensor Networks“. In : *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 8* -. T. 8. HICSS '00. IEEE Computer Society, 2000 (cf. p. 29).

[Hei+02] Wendi B. HEINZELMAN, Ananthan P. CHANDRAKASAN et Hari BALAKRISHNAN. „An application-specific protocol architecture for wireless microsensor networks“. In : *IEEE Transactions on Wireless Communications* 1.4 (2002), p. 660–670 (cf. p. 32).

[Iee] *IEEE Standard for Low-Rate Wireless Networks, IEEE Std 802.15.4*. Rapp. tech. IEEE, 2015 (cf. p. 14, 18).

[Int+00] Chalermek INTANAGONWIWAT, Ramesh GOVINDAN et Deborah ESTRIN. „Directed Diffusion : A Scalable and Robust Communication Paradigm for Sensor Networks“. In : *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*. MobiCom '00. ACM, 2000, p. 56–67 (cf. p. 64).

[Iva+07] Stoianov IVAN, Nachman LAMA et Madden SAM. „PIPENET : A Wireless Sensor Network for Pipeline Monitoring“. In : *International Conference on Information Processing in Sensor Networks (IPSN 2007)* (2007), 264–273 (cf. p. 12).

[Jar30] VOJTĚCH JARNÍK. „ O Jistém Problému Minimálním (About a Certain Minimal Problem) (in Czech, German summary)“. In : *Práce Mor. Prírodoved. Spol. v Brne VI* 4 (1930), p. 57–63 (cf. p. 70, 75).

[Jaw+07] Imad JAWHAR, Nader MOHAMED et Khaled SHUAIB. „A framework for pipeline infrastructure monitoring using wireless sensor networks“. In : *Wireless Telecommunications Symposium, 2007. WTS 2007*. IEEE, 2007, 1–7 (cf. p. 1, 12).

[Jaw+09] Imad JAWHAR, Nader MOHAMED, Khaled SHUAIB et Nader KESSERWAN. „An Efficient Framework and Networking Protocol for Linear Wireless Sensor Networks.“ In : *Ad Hoc & Sensor Wireless Networks* 7 (2009) (cf. p. 59, 60, 67).

[Jaw+11] Imad JAWHAR, Nader MOHAMED et Dharma P. AGRAWAL. „Linear wireless sensor networks : Classification and applications“. In : *Journal of Network and Computer Applications* 34.5 (2011), p. 1671–1682 (cf. p. 3, 5, 35, 38, 39, 59, 70).

[Jeo+09] Ko JEONGGIL, Gao TIA et Terzis ANDREAS. „Empirical study of a medical sensor application in an urban emergency department“. In : *4th International ICST Conference on Body Area Networks, BODYNETS 2009*. ICST, 2009, p. 10 (cf. p. 10).

[Jeo+10] Ko JEONGGIL, Lim JONG HYUN, Chen YIN et al. „MEDiSN : Medical emergency detection in sensor networks“. In : *ACM Trans. Embedded Comput. Syst.* 10 (2010) (cf. p. 10).

[Jie+02] Wu JIE, Dai FEI, Gao MING et Stojmenovic IVAN. „On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks“. In : *Journal of Communications and Networks* 4.1 (2002), p. 59–70 (cf. p. 28).

[Jua+02] Philo JUANG, Hidekazu OKI, Yong WANG et al. „Energy-efficient Computing for Wildlife Tracking : Design Tradeoffs and Early Experiences with ZebraNet“. In : *SIGARCH Comput. Archit. News* 30.5 (2002), p. 96–107 (cf. p. 11).

[Jun+10] Wang JUN, Zhang XIN, Xie JUNYUAN et Mi ZHENGKUN. „A distance-based clustering routing protocol in wireless sensor networks“. In : *2010 IEEE 12th International Conference on Communication Technology*. 2010, p. 648–651 (cf. p. 31).

[Jun+11] Young Jin JUNG, Yang Koo LEE, Dong Gyu LEE et al. „Design of Sensor Data Processing Steps in an Air Pollution Monitoring System“. In : *Sensors* 11.12 (2011), p. 11235–11250 (cf. p. 10).

[Kar+09] Theodora KARVELI, Konstantinos VOULGARIS, Mohammad GHAVAMI et A. Hamid AGHVAMI. „DiS-MAC : A MAC protocol for sensor networks used for roadside and highway monitoring“. In : *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*. IEEE, 2009, p. 1–6 (cf. p. 41, 45).

[Kim+07] Sukun KIM, Shamim PAKZAD, David CULLER et al. „Health monitoring of civil infrastructures using wireless sensor networks“. In : *Proceedings of the 6th international conference on Information processing in sensor networks*. ACM Press, 2007, p. 254–263 (cf. p. 1, 3, 12).

[KJ11] Hyung Seok KIM et Yoon JIWON. „Hybrid Distributed Stochastic Addressing Scheme for ZigBee/IEEE 802.15.4 Wireless Sensor Networks“. In : *ETRI Journal* 33.5 (2011), p. 704–711 (cf. p. 4, 5).

[KT05] Jon KLEINBERG et Eva TARDOS. *Algorithm Design*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2005 (cf. p. 75).

[Lim+10] Yujin LIM, Hak-Man KIM et Sanggil KANG. „A Design of Wireless Sensor Networks for a Power Quality Monitoring System“. In : 10 (nov. 2010), p. 9712–25 (cf. p. 13).

[LJY01] Li LI et Halpern JOSEPH Y. „Minimum-energy mobile wireless networks revisited“. In : *IEEE International Conference on Communications*. T. 1. 2001, p. 278–283 (cf. p. 26, 27).

[Lor+06] Konrad LORINCZ, Matt WELSH, Omar MARCILLO et al. „Deploying a wireless sensor network on an active volcano“. In : *IEEE Internet Computing*. 2006, p. 18–25 (cf. p. 1, 11).

[Lrw] „IEEE Standard for Low-Rate Wireless Networks“. In : *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (2016), p. 1–709 (cf. p. 13).

[Ma+08] Y MA, M RICHARDS, M GHANEM, Y GUO et J HASSARD. „Air pollution monitoring and mining based on sensor Grid in London“. In : *SENSORS* 8 (2008), p. 3601–3623 (cf. p. 10, 11).

[Mao+12] Xufei MAO, Xin MIAO, Yuan HE, Xiang-Yang LI et Yunhao LIU. „CitySee : Urban CO 2 monitoring with sensors“. In : *INFOCOM, 2012 Proceedings IEEE*. 2012, p. 1611–1619 (cf. p. 10).

[Mig+15] M.E. MIGABO, K. DJOUANI, A.M. KURIEN et T.O. OLWAL. „Gradient-based Routing for Energy Consumption Balance in Multiple Sinks-based Wireless Sensor Networks“. In : *Procedia Computer Science* 63 (2015), p. 488 –493 (cf. p. 61, 64).

[Moo+10] Shim MOONJOO, Kim DONGKYUN, Yoo HONGSEOK et Hyung Kim KYU. „GLOBAL : A Gradient-based routing protocol for load-balancing in large-scale wireless sensor networks with multiple sinks“. In : *2010 IEEE Symposium on Computers and Communications (ISCC)* (2010), p. 556–562 (cf. p. 61, 65).

[MP11] Luca MOTTOLA et Gian Pietro PICCO. „MUSTER : Adaptive Energy-Aware Multisink Routing in Wireless Sensor Networks“. In : *IEEE Transactions on Mobile Computing* 10.12 (2011) (cf. p. 61).

[Nbi] *NB-IoT*. 2017. URL : `http://www.3gpp.org` (cf. p. 20).

[Ndo+14a] E. H. M. NDOYE, F. JACQUET, M. MISSON et I. NIANG. „Using a token approach for the MAC layer of linear sensor networks : Impact of the node position on the packet delivery“. In : *2014 IFIP Wireless Days (WD)*. 2014, p. 1–4 (cf. p. 38, 70).

[Ndo+14b] El Hadji Malick NDOYE, Frédérique JACQUET, Michel MISSON et Ibrahima NIANG. „Using a token approach for the MAC layer of linear sensor networks : Impact of the node position on the packet delivery“. In : *2014 IFIP Wireless Days, WD 2014, Rio de Janeiro, Brazil, November 12-14, 2014*. 2014, p. 1–4 (cf. p. 145).

[Ndo+16] El Hadji Malick NDOYE, Ibrahima NIANG, Frédérique JACQUET et Michel MISSON. „LTB-MAC : Linear Token-Based MAC Protocol for linear sensor network“. In : *Proceedings of CARI 2016*. 2016 (cf. p. 41, 47, 66, 151).

[Ndo15] EL hadji Malick NDOYE. „Réseaux de capteurs sans linéaires : impact de la connectivité et des interférences sur une méthode d'accès basée sur des jetons circulant“. EDSPI, Université Blaise Pascal, 2015 (cf. p. 3).

[Pan+08] Meng-Shiuan PAN, Hua-Wei FANG, Yung-Chih LIU et Yu-Chee TSENG. „Address Assignment and Routing Schemes for ZigBee-Based Long-Thin Wireless Sensor Networks“. In : *IEEE Vehicular Technology Conference '08, Spring 2008*. 2008, p. 173–177 (cf. p. 1, 3–5, 35, 40, 56, 66, 67).

[Pan+09] Meng-Shiuan PAN, Chia-Hung TSAI et Yu-Chee TSENG. „The Orphan Problem in ZigBee Wireless Networks“. In : *IEEE Transactions on Mobile Computing* 8 (2009) (cf. p. 51).

[Par+09] Sungjin PARK, Eun Ju LEE, Jae Hong RYU, Seong-Soon JOO et Hyung Seok KIM. „Distributed Borrowing Addressing Scheme for ZigBee/IEEE 802.15.4 Wireless Sensor Networks“. In : *ETRI journal* 31.5 (2009) (cf. p. 4, 5, 52, 121).

[Per+03] C. PERKINS, E. ROYER et S. DAS. *RFC 3561 Ad hoc On-Demand Distance Vector (AODV) Routing*. Rapp. tech. 2003 (cf. p. 48).

[Pri57] R. C. PRIM. „Shortest connection networks and some generalizations“. In : *Bell System Technology Journal* 36 (1957), p. 1389–1401 (cf. p. 70, 75).

[Raz+17] Usman RAZA, Parag KULKARNI et Mahesh SOORIYABANDARA. „Low Power Wide Area Networks : An Overview“. In : *IEEE Communications Surveys Tutorials* 19.2 (2017), p. 855–873 (cf. p. 20).

[RR16] Bushra RASHID et Mubashir Husain REHMANI. „Applications of wireless sensor networks for urban areas : A survey“. In : *Journal of Network and Computer Applications* 60 (2016), p. 192–219 (cf. p. 9).

[San05] Paolo SANTI. „Topology control in wireless ad hoc and sensor networks“. In : *ACM computing surveys (CSUR)* 37.2 (2005), p. 164–194 (cf. p. 26).

[SD15] Radosveta SOKULLU et Eren DEMIR. „A Comparative Study of MAC protocols for Linear WSNs“. In : *Procedia Computer Science* 52 (2015), p. 492 –499 (cf. p. 145).

[See+11] C. SEE, K. HOROSHENKOV, R. abd ALHMEED, Y. HU et S. TAIT. „A Low Power Wireless Sensor Network for Gully Pot Monitoring in Urban Catchments“. In : *IEEE Sensors Journal* (2011) (cf. p. 12).

[Sha+07] G. M. SHAFIULLAH, Amoakoh GYASI-AGYEI et Peter WOLFS. „Survey of wireless communications applications in the railway industry“. In : *Wireless Broadband and Ultra Wideband Communications, 2007. AusWireless 2007. The 2nd International Conference on*. IEEE, 2007, p. 65–65 (cf. p. 12).

[Sig] *Sigfox*. 2017. URL : `https://www.sigfox.com` (cf. p. 20).

[Sik+06] Pavan SIKKA, Peter CORKE, Philip VALENCIA et al. „Wireless adhoc sensor and actuator networks on the farm“. In : *Proceedings of the 5th international conference on Information processing in sensor networks*. ACM, 2006, p. 492–499 (cf. p. 11).

[Soc11] I.E.E.E. Computer SOCIETY. *Part 15.4 : Low-Rate Wirelees Personal Area Networks (LR-WPANs)*. IEEE. IEEE Computer Society. ANSI/IEEE, 2011 (cf. p. 14, 18).

[Sor+16] N. SORNIM, M. LUIS, T. EIRICH et T. KRAMP. *LoRaWAN Specification*. LoRa Alliance. ZigBee, 2016 (cf. p. 20, 21).

[Tan+12] Qiang TANG, Kun YANG, Ping LI et al. „energy efficient MCDS construction algorithm for wireless sensor networks“. In : *EURASIP Journal on Wireless Communications and Networking* 2012 (2012), p. 83 (cf. p. 26).

[Tse+10] Yuriy TSELISHCHEV, Athanassios BOULIS et Libman LIBMAN. „Experiences and Lessons from Implementing a Wireless Sensor Network MAC Protocol in the Castalia Simulator“. In : *Wireless Communications and Networking Conference (WCNC), 2010 IEEE*. IEEE, 2010, p. 1–6 (cf. p. 106).

[VTH99] Rodoplu VOKAN et Meng TERESA H. „Minimum energy mobile wireless networks“. In : *IEEE Journal on Selected Areas in Communications* 17.8 (1999), p. 1333–1344 (cf. p. 26).

[Wan+11] Jiangwen WAN, Yang YU, Yinfeng WU, Renjian FENG et Ning YU. „Hierarchical Leak Detection and Localization Method in Natural Gas Pipeline Monitoring Sensor Networks“. In : *Sensors* 12.12 (2011-12-27), p. 189–214 (cf. p. 12).

[Was] *Waspmote sigfox networking guide*. 2017. URL : `http://www.libelium.com/downloads/` (cf. p. 20, 21).

[Wat+06] Thomas WATTEYNE, Isabelle AUGÉ-BLUM et Stéphane UBÉDA. „Dual-mode real-time mac protocol for wireless sensor networks : a validation/simulation approach“. In : *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*. ACM, 2006, p. 2 (cf. p. 41, 43, 66).

[Ye+02] Wei YE, Jhon S. HEIDEMANN et Deborah ESTRIN. „An energy-efficient MAC protocol for wireless sensor networks“. In : *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*. T. 3. 2002, 1567–1576 vol.3 (cf. p. 107).

[Yic+08] Jennifer YICK, Biswanath MUKHERJEE et Dipak GHOSAL. „Wireless sensor network survey“. In : *Comput. Netw.* 52.12 (2008), p. 2292–2330 (cf. p. 2, 9).

[Yoo+11] Sunhee YOON, Wei YE, John HEIDEMANN, Brian LITTLEFIELD et Cyrus SHAHABI. „SWATS : Wireless sensor networks for steamflood and waterflood pipeline monitoring“. In : *IEEE network* 25.1 (2011) (cf. p. 12).

[Yua+06a] Zeng YUANYUAN, Xiaohua JIA et He YANXIANG. „Energy efficient distributed connected dominating sets construction in wireless sensor networks“. In : *Proceedings of the 2006 international conference on Wireless communications and mobile computing*. ACM, 2006, p. 797–802 (cf. p. 26).

[Yua+06b] Zeng YUANYUAN, Xiaohua JIA et He YANXIANG. „Energy Efficient Distributed Connected Dominating Sets Construction in Wireless Sensor Networks“. In : *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing*. ACM, 2006, p. 797–802 (cf. p. 29).

[Zar+11] Manuel ZARZO, Angel FERNANDEZ-NAVAJAS et Fernando-Juan GARCIA-DIEGO. „Long-Term Monitoring of Fresco Paintings in the Cathedral of Valencia (Spain) Through Humidity and Temperature Sensors in Various Locations for Preventive Conservation“. In : *Sensors* 11.12 (2011), p. 8685–8710 (cf. p. 12).

[Zim+08] Marco ZIMMERLING, Waltenegus DARGIE et Johnathan M. REASON. „Localized power-aware routing in linear wireless sensor networks“. In : *Proceedings of the 2nd ACM international conference on Context-awareness for self-managing systems*. CASEMANS '08. New York, NY, USA : ACM, 2008, 24–33 (cf. p. 1, 3, 35).

# Résumé

Les réseaux de capteurs sans fils linéaires (RdCSL) sont un cas particulier de réseaux de capteurs sans fils où les nœuds de capteurs sont déployés le long de multiples lignes. les RdCSL sont utilisés pour la surveillance des infrastructures routières, ferroviaires, des conduites de gaz, d'eau, de pétrole et de cours d'eau.
Les solutions classiques de formation de topologie et d'adressage proposées ne sont pas adaptées à l'environnement des RdCSFL. En effet les paramètres initiaux utilisés par ces protocoles tels que le nombre maximum de nœuds fils ($Cm$), nombre maximum de nœuds routeurs fils $Rm$, profondeur maximum de l'arbre ($Lm$), occasionnent un gaspillage de l'espace d'adressage disponible pour les nœuds et limitent la profondeur de l'arbre adressable ($15$ sauts pour ZigBee). D'autres solutions adaptées pour les RdCSFL utilisent une organisation en cluster des nœuds du réseau et sont basées elles aussi sur des paramètres fixés à l'avance tels quel le nombre maximum de cluster fils par cluster. De plus, ces solutions requièrent beaucoup d'interventions manuelles sur les nœuds de capteurs (choix des chefs de cluster par exemple) et ne favorisent pas une adaptation face aux changements du RdCSL tels que l'ajout d'un ensemble de nœuds de capteurs.
Dans cette thèse, nous proposons donc des protocoles permettant la construction automatique de topologies logiques, l'adressage et le routage pour des réseaux de capteurs sans fil linéaires. Nos protocoles fournissent aussi des mécanismes de gestion dynamique d'un RdSFL avec l'ajout de nouveaux nœuds, la réallocation d'adresses pour les nœuds en cas d'épuisement de blocs d'adresses et la gestion du routage vers plusieurs puits du réseau.
Nos différents protocoles sont évalués grâce au simulateur Castalia/Omnet++. Les résultats de nos simulations montrent que nos protocoles permettent de construire un RdCSFL connecté avec peu de nœuds orphelins (nœuds sans adresses logiques) et sans limitations de profondeur. Nous montrons aussi, grâce à nos simulations, que nos contributions permettent d'ajouter un grand nombres de nœuds à un RdCSFL existant de n'importe quelle taille et s'adaptent au déploiement de plusieurs puits et au routage multi-puits et permettent d'améliorer le ratio et la latence de paquets livrés dans les RdCSFL.
**Mots-clés** : Réseaux de capteurs sans fil linéaires, construction de topologies, déploiement automatique, adressage, routage, multi-puits

# Abstract

Linear wireless sensor network (LWSN) are a sub-case of wireless sensor network where sensor nodes are roughly deployed through multiple long lines with branches. LWSN are used to monitor infrastructures such as roads, pipelines, and naturals entities such as rivers.
Classical solutions of topology construction and addressing are inefficient on LWSN . Indeed, with initials networks parameters such as the maximum number of children per node ($Cm$), the maximum number of children routers per node ($Rm$), and the maximum tree depth, a solution like ZigBee causes a waste of available address space of network nodes and limit the depth of the addressable tree to $15$ hops. Other solutions proposed for LWSN use a cluster-tree organisation and are based on initial network parameters such as the maximum number of children clusters per cluster. In addition, these solutions require a lot of manual intervention on different sensor nodes and do not allow adaptation for a network extension (addition of a set of new sensor nodes).
In this thesis, we propose protocols to allow the automatic construction of topologies, the addressing and the data routing for linear wireless sensor networks. Our contribution also provides mechanisms for dynamic management of LWSN (addition of new nodes, addresses reallocation, and data routing to multiple sink nodes).
Our different protocols are evaluated using Castalia/Omnet++ simulator. Results of our simulations show that our protocols allow a construction of connected LWSN with very few orphan nodes and without depth limitations. We also show that our contribution allows to add many new nodes on different LWSN, and adapts to the deployment of multiple sinks to improve the ratio and the latency of data delivery packets.
**Keywords** : Linear wireless sensor network, topology construction, addressiing, routing, multi-sink