Université de Reims Champagne-Ardenne
École Doctorale Sciences Technologies et Santé

Thèse

présentée par **Bachar Salim HAGGAR**

pour l'obtention du grade de

Docteur de l'Université de Reims Champagne-Ardenne

Spécialité : Informatique

Auto-organisation et routage dans les réseaux mobiles ad hoc

Président	Hervé Guyennet	Professeur à l'Université de Besançon
Rapporteur	Marc Bui	Professeur et Directeur d'Études Cumulant à l'EPHE
Rapporteur	Mohamed Mosbah	Professeur à l'Institut Polytechnique de Bordeaux
Examinateur	Michaël Krajecki	Professeur à l'Université de Reims Champagne-Ardenne
Examinateur	Ibrahima Niang	Enseignant-Chercheur et Directeur du Centre de Calcul de l'UCAD
Directeur	Olivier Flauzac	Professeur à l'Université de Reims Champagne-Ardenne
Co-directeur	Florent Nolot	Maître de Conférences à l'Université de Reims Champagne-Ardenne

---

Centre de Recherche en STIC équipe Systèmes Communicants

“À la mémoire de mon père”
“À ma mère”
“À mes frères et sœurs”
“À ma famille”
“À tous ceux qui me sont chers”

Remerciements

Je tiens à exprimer mes remerciements et toute ma gratitude à mon directeur *Olivier Flauzac* et à mon co-directeur *Florent Nolot* respectivement Professeur et Maître de conférences à l'Université de Reims Champagne-Ardenne pour avoir accepté d'encadrer cette thèse. J'aimerais leur adresser mes plus vifs remerciements pour tout leur dynamisme, leur tolérance, leur disponibilité, leur compétences scientifiques que j'ai pu apprécier tout au long de ces années. Ce travail n'aurait jamais pu aboutir sans eux qui ont toujours su me consacrer des moments de leur temps et me témoigner leur soutien et leur confiance. Je souhaite leur transmettre l'expression de ma plus profonde et sincère gratitude.

Une pensée toute particulière pour *Cyril Rabat*, Maître de Conférences à l'Université de Reims Champagne-Ardenne, dont le dynamisme m'a particulièrement encouragé dans la dernière ligne droite. Sans ses exceptionnelles idées et sa disponibilité pour m'aider à mener à bien la partie expérimentale et la rédaction, ce travail n'aurait jamais pu aboutir. J'ai beaucoup appris en travaillant avec lui et je le remercie de tout mon cœur.

Je remercie aussi mes rapporteurs *Mohamed Mosbah*, Professeur à l'Institut Polytechnique de Bordeaux et *Marc Bui*, Professeur et Directeur d'Études Cumulant à l'EPHE, pour avoir rapporté mes travaux. Leurs remarques pertinentes sur une première version de ce document m'ont permis d'en améliorer la clarté et de considérer des nouvelles perspectives de mon travail.

Je remercie le Professeur *Herve Guyennet* de l'Université de Besançon qui m'a fait l'honneur d'accepter de participer au jury.

Je remercie le Professeur *Michaël Krajecki* de l'Université de Reims Champagne-Ardenne qui m'a fait l'honneur d'accepter de participer au jury et qui m'a offert un excellent cadre de travail ainsi qu'un séjour extrêmement agréable.

Je tiens également à remercier *Ibrahima Niang* enseignant-chercheur et directeur du centre de calcul de l'UCAD, qui m'a fait l'honneur d'accepter de participer au jury.

Je remercie mon collègue de bureau *Arnaud Renard* qui m'a supporté toutes ces années, pour tous ses conseils, pour toute son aide au niveau organisationnel, pour sa grande gentillesse, pour ses encouragements et aussi pour sa bonne humeur quotidienne.

Je remercie également l'ensemble des enseignants, l'ensemble du personnel et doctorants avec qui j'ai passé des moments exceptionnels. Merci à *Luiz Angelo Steffenel*, *Christophe Jaillet*, *Thibault Bernard*, *Jean-Charles Boisson*, *Hacène Fouchal*, *Pierre Delisle*, *Hervé Deleau*,

Hichem Baala, *Kudireti Abdurusul*, *Gabriel Noaje*, *Audrey Delevacq*, *Mandicou Ba*, *Marie Carlier*, *Iyad Alshabani*, *Sylvain Darras*, *Sergine Bristiel* et *Christophe Goessen*.

Mes remerciements vont également à tous ceux qui ont contribué de près ou de loin à l'élaboration de cette thèse, qu'ils trouvent ici l'expression de ma profonde sympathie.

En plus des personnes avec lesquelles j'ai pu coopérer sur ce travail, je remercie celles qui ont assisté à ma soutenance.

Mes derniers remerciements vont à ma famille et notamment à ma mère, mes frères, mes sœurs, mes amis et tous ceux qui me sont chers, car sans leur constant soutien et leurs encouragements, je n'aurais pas pu mener à bien cette thèse. Cette page serait loin de suffire pour vous exprimer toute ma reconnaissance et mon affection.

Merci infiniment encore à tous.

Table des matières

Table des figures

Liste des algorithmes

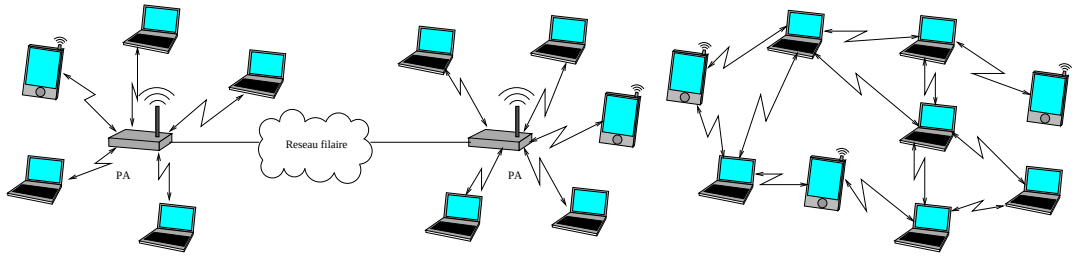
Introduction

Au cours de ces dernières années, les réseaux mobiles sans fil ont connu un très fort développement pour répondre à la hausse constante des besoins en mobilité. L'exemple le plus significatif est l'émergence des appareils de communications tels que : les téléphones cellulaires, les PDA [1], les ordinateurs portables mais aussi le large déploiement des réseaux locaux sans fils avec la multiplication des points d'accès dans les lieux publics et chez les particuliers. Ainsi, avec le développement de ces réseaux et la montée en puissance du haut débit, l'utilisateur a accès à l'information en tous lieux et à tout moment. Un utilisateur en déplacement peut accéder à sa messagerie, gérer ses mails, naviguer sur internet dans un train, dans un hall d'aéroport, ou dans un hôtel. Dans une conférence, les participants peuvent échanger des fichiers grâce à leurs ordinateurs portables via un réseau local sans fil, dans un campus, des étudiants peuvent échanger des fichiers. Ainsi, ces réseaux sont en plein développement du fait de leur flexibilité de leur interface, qui offre à l'utilisateur la mobilité.

Les réseaux ad hoc représentent une composante clé de cette évolution. Ces réseaux s'organisent automatiquement de façon à être déployés rapidement en permettant des échanges directs entre stations mobiles. Ainsi, le fonctionnement du réseau repose sur les stations elles-mêmes : celles-ci interviennent à la fois comme terminaux pour communiquer avec les usagers et comme routeurs afin de relayer le trafic pour le compte d'autres utilisateurs (voir figure 1(b)). Par conséquent, l'envoi d'un paquet n'est plus pris en charge par des équipements tels que *routeurs* ou *commutateurs*. De plus, le réseau doit être dynamique : les nœuds du réseau peuvent se déplacer de façon libre et arbitraire. Ainsi, un nœud peut quitter ou rejoindre le réseau à tout instant. Les nœuds peuvent communiquer dans la limite de la portée de leur communication radio. Lorsqu'un émetteur ne peut communiquer directement avec le destinataire, les informations devront être transmises de proche en proche avant d'atteindre la destination souhaitée. Cette communication est assurée par un protocole de routage. Le protocole doit prendre en compte les caractéristique propres du réseau (variabilité de la topologie, absence d'infrastructure), l'hétérogénéité des nœuds (différences en termes de CPU, mémoire) et la bande passante limitée.

Les réseaux ad hoc peuvent également être connectés aux réseaux filaires ou internet via des passerelles, que nous appellerons des point d'accès (AP) (voir figure 1(a)). De tels réseaux sont communément appelés réseaux cellulaires. Dans ce cas, le réseau ad hoc élargit l'accès aux services du réseau filaire.

1. Personal Digital Assistant



(a) réseau cellulaire (b) réseau ad hoc

FIGURE 1 – Comparaison entre un réseau cellulaire et un réseau ad hoc

Avec l'émergence de ces dispositifs, de multiples recherches ont été faites dans des communications mobiles. Ces recherches se concentrent en particulier sur les façons de router les informations entre les nœuds du réseau. Il existe aujourd'hui de nombreux protocoles de routage pour de tels réseaux. Ainsi, les premiers protocoles proposés sont des protocoles de routage à *plat*. Cette approche considère que tous les nœuds du réseau sont dans le même niveau de hiérarchie et possède ainsi les mêmes rôles. Bien qu'efficace pour des réseaux de petite taille, ils ne permettent pas le passage à l'échelle quand le nombre de nœuds du réseau augmente. Par la suite, des protocoles de routage *hiérarchiques* sont apparus. Cette approche consiste à découper le réseau en groupes d'entités appelés *clusters* en donnant au réseau une structure hiérarchique et utilise des schémas de routage différents entre les clusters et au sein des clusters. Dans cette approche, chaque nœud maintient une connaissance proactive au sein de son cluster et seulement une connaissance partielle pour les autres clusters. Bien que ces protocoles permettent le passage à l'échelle, ils subissent cependant continuellement des changements de topologie.

Pour ces multiples raisons, il nous a semblé intéressant d'étudier plus en avant le mécanisme permettant au réseau de s'adapter automatiquement aux changements topologiques. Les algorithmes auto-stabilisants se sont révélés être des solutions plus réalistes face aux problèmes de pannes transitoires dans le réseau. Dans ce manuscrit, nous utilisons le paradigme d'auto-stabilisation pour traiter les défaillances pouvant survenir dans le réseau.

Nous présentons tout d'abord deux solutions auto-stabilisantes, l'une pour découper le réseau en clusters et l'autre pour construire un arbre couvrant du réseau. Le découpage du réseau en clusters permet de rendre l'utilisation du réseau plus efficace, par exemple le routage, la diffusion, etc. La structure d'arbre couvrant permet de faire une diffusion d'informations dans le réseau. La troisième solution s'intéresse à la construction d'un protocole de routage basé sur une structure hiérarchique.

Organisation de la thèse

Cette thèse est organisée comme suit :

Dans le premier chapitre, nous définirons les principaux concepts liés aux systèmes distribués, puis nous introduirons les différentes approches permettant à un système distribué de

garantir le bon fonctionnement du système malgré les dysfonctionnements des différents composants du système. Nous présenterons le concept de système auto-stabilisant permettant de tolérer des fautes transitoires.

Dans le deuxième chapitre, nous présenterons les environnements mobiles tout en décrivant de manière synthétique les réseaux avec infrastructure et réseaux ad hoc. Nous mettons ensuite en évidence les caractéristiques des réseaux ad hoc afin d'identifier les défis posés par ces réseaux. La variabilité de la topologie entraîne par exemple la réorganisation du réseau. L'absence d'infrastructure oblige les nœuds à s'auto-organiser et à s'auto-gérer.

Dans le chapitre 3, nous présenterons les différentes approches proposées dans la littérature pour structurer le réseau en clusters. Nous présenterons les deux catégories des algorithmes de clustering : les algorithmes de clustering à 1 saut et les algorithmes de clustering à k sauts. Cette étude relève les limites des approches actuelles et nous sert de support pour positionner nos travaux de recherche. Nous présenterons ensuite la structure hiérarchique que nous proposons. L'algorithme que nous proposons est basé sur des connaissances locales pour construire les clusters. Dans ce chapitre, nous montrons également que notre algorithme est auto-stabilisant, cette propriété lui permet de tolérer les fautes transitoires. A partir de cette structure en clusters, nous proposons deux applications : une diffusion efficace d'information dans le réseau et un protocole de routage. Un ensemble de simulations, sous différents scénarios est réalisé à l'aide de l'outil de simulation $DASOR^2$[Rab]. Les résultats obtenus montrent la pertinence de l'organisation proposée.

Dans le chapitre 4, nous présenterons les différentes approches de diffusion d'information proposées dans la littérature. Nous présenterons ensuite notre protocole de diffusion. Ce protocole construit un arbre couvrant inter-clusters. Il permet de faire une diffusion efficace dans le réseau. De plus, la construction de la structure d'arbre couvrant ne requiert pas une connaissance globale du réseau. Ainsi, la construction de l'arbre et la diffusion d'information ne demande que peu d'échanges de messages.

Dans le chapitre 5, nous présenterons trois solutions de routage tirant parti de la structure hiérarchique introduite dans le réseau. Cette structure permet d'utiliser deux modes de routage différents : un protocole de routage proactif au sein des clusters et un protocole de routage hybride entre les clusters. L'objectif de ces solutions est à la fois d'optimiser le nombre de messages échangés et le délai de bout en bout.

Enfin le dernier chapitre conclura cette thèse. Nous exposerons également les perspectives concernant les travaux traités dans ce document.

2. Distributed Algorithm Simulator Of Reims University

Chapitre 1

Les systèmes distribués

Résumé : *Ce chapitre propose un aperçu sur les systèmes distribués. Un système distribué est généralement représenté sous forme de graphes. Nous décrivons les topologies les plus couramment utilisées et nous donnons quelques définitions sur les différents concepts liés aux systèmes distribués (élection, exclusion mutuelle, synchronisation, etc.). Dans la deuxième partie, nous introduisons les différentes approches permettant à système distribué de garantir un bon fonctionnement malgré les éventuels dysfonctionnements qui peuvent survenir. À la fin de ce chapitre, nous présentons l'approche d'auto-stabilisation permettant à un système de résister aux fautes transitoires. Cette approche est utilisés tout au long de cette thèse pour répondre les principaux problèmes classiques de l'algorithmique distribué.*

1.1 Systèmes distribués

Les systèmes distribués sont des systèmes qui gèrent des unités de traitement qui sont aussi appelés *processus*. Les différentes unités de traitement communiquent entre elles en échangeant des informations appelés *messages*, par l'intermédiaire de *canaux* de communication. Plusieurs systèmes peuvent être considérés comme des systèmes distribués. Ainsi sur un système d'exploitation multi-processus, plusieurs peuvent fonctionner simultanément et communiquer entre eux. Dans un ordinateur multi-processeurs, les processeurs peuvent communiquer entre eux, comme ceux d'ordinateurs mis en réseau.

De nos jours, avec la multiplication des ordinateurs en réseau, notamment dû à la démocratisation et la facilité d'accès à internet, l'étude des systèmes distribués prend tout son essor. Avec la croissance de la puissance des processeurs, des calculs de plus en plus complexes peuvent être effectués. Mais afin de pouvoir utiliser plusieurs machines pour solutionner un problème donné, il faut encore avoir des programmes qui le permettent. L'algorithmique distribuée est l'étude de ces problèmes. Un algorithme distribuée est constitué de l'ensemble des algorithmes des processeurs du système. L'algorithme de chaque processus est constitué d'actions qui sont soit la réception de messages, soit l'émission d'une information vers un autre processus, soit une instruction interne. Mais il n'est pas si simple de concevoir des algorithmes distribués performants, c'est à dire qu'ils permettent d'obtenir le résultat voulu en un minimum de temps, effectuant un minimum d'échange d'information et utilisant peu d'espace mémoire. Une comparaison entre un système centralisé et distribué permet de résumer les difficultés rencontrées en trois points :

1. Localité des informations : dans un système centralisé, un processus peut connaître les valeurs de toutes les variables (ou états) utilisés par tous les autres processus. Dans un système distribué, un processus n'a la possibilité de connaître que les états qui lui sont envoyés par les processus auxquels il est directement connecté (aussi appelé *voisins*). Cependant ces informations peuvent avoir été modifiées pendant le temps de la transmission et donc être obsolètes lorsque le processus destinataire en prendra connaissance,
2. Localité de temps : dans un système centralisé, les exécutions se font séquentiellement. Dans un système distribué, chaque processus exécute des actions selon une relation d'ordre partiel et non total. D'une manière générale, entre deux processus exécutés en parallèle, nous ne savons donc pas lequel se terminera en premier,
3. Tous les composants mis en jeu dans un système distribué ne fonctionnent pas forcément à la même vitesse. Il est donc impossible de déterminer à l'avance le comportement global d'une suite de processus (d'un algorithme distribué).

1.1.1 Les différents systèmes

Nous pouvons classer les systèmes distribués suivant différentes hypothèses, des plus particulières aux plus générales. Les algorithmes distribués sont donc écrits pour fonctionner sous certaines de ses hypothèses. Évidemment un algorithme écrit sous certaines hypothèses fonctionnera sous des hypothèses plus restrictives alors que l'inverse ne pourra être vrai sans l'adjonction d'un sous système (si c'est possible) chargé de transformer les hypothèses. Nous pouvons ainsi différentier les systèmes suivant plusieurs critères.

- La topologie : les différentes topologies sont représentées par un graphe sur lequel les arrêtes représentent les liens de communication et les sommets les processus. Les différentes topologies couramment utilisées sont les *chaînes* (figure 1.1(a), *étoiles* (figure 1.1(c)), *anneaux* (figure 1.1(b)), *grilles* (figure 1.1(e)), *arbres* (figure 1.1(d)) ou *cliques* (figure 1.1(f)),
- Les communications : les liens de communication peuvent être soit bidirectionnels, soit unidirectionnels, c'est à dire les processus échangent des données dans les deux sens ou bien dans un seul sens. Ainsi si un processus P_1 envoie des données à P_2 par un lien unidirectionnel alors P_2 ne pourra pas envoyer d'informations à P_1 par ce même lien. Avec des liens unidirectionnels, il existe des topologies particulières telles que l'anneau unidirectionnel (figure 1.2(a)) ou plus généralement le graphe fortement connexe (figure 1.2(b)) où chaque processus peut communiquer avec tous les autres.

Il existe plusieurs hypothèses possibles sur les communications, de la *communication globale* à la communication point à point, en passant par la *communication multi-points*. Dans une *communication globale*, un processus peut communiquer avec tous les autres. Dans une *communication multi-points*, un sous ensemble de processus peuvent communiquer ensemble alors que dans la communication *communication point-à-point*, seulement deux processus peuvent s'échanger des informations grâce à un lien de communication qui les relie.

- Synchronisme : il existe deux types de systèmes : les systèmes *synchrones* et *asynchrones*. Dans un système synchrone, tous les processus exécutent une action exactement en même temps. Dans un système asynchrone, chaque composant fonctionne à sa propre vitesse,
- Connaissances globales : la première hypothèse est liée aux identifiants que peuvent avoir les sommets. Chaque sommet peut avoir un identifiant unique afin de pouvoir tous les différentier. Si tous les processus sont identiques avec aucun moyen de les différentier,

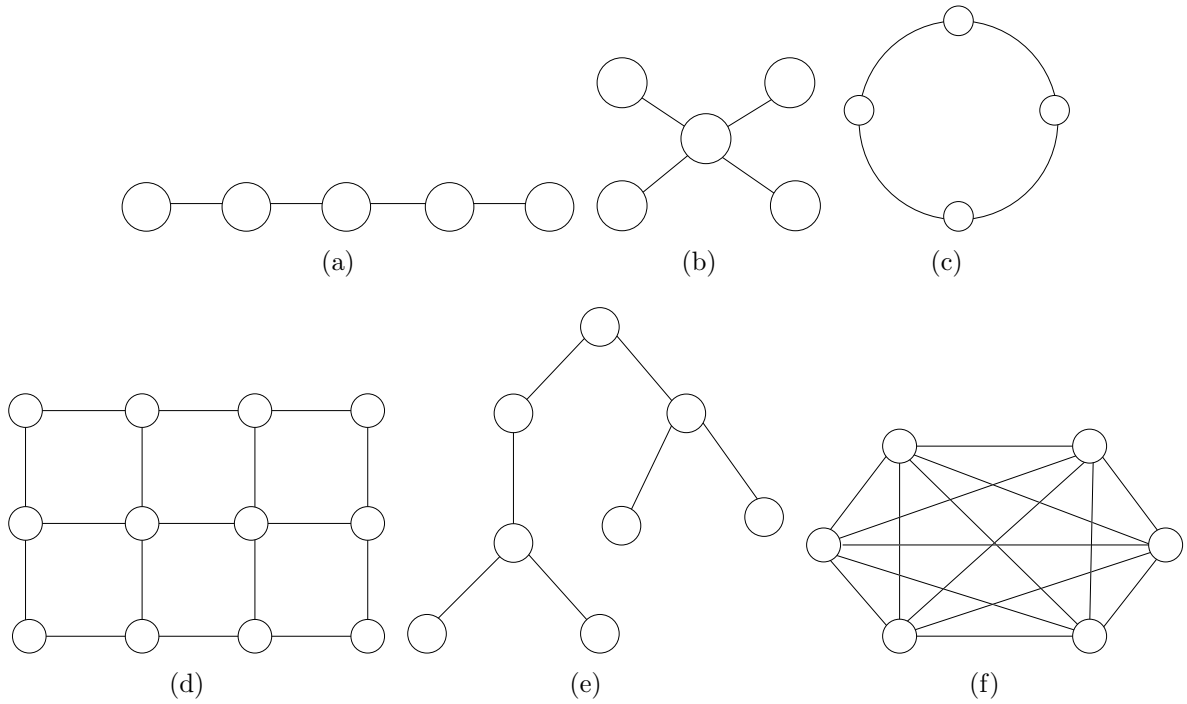


FIGURE 1.1 – Quelques topologies non orientées usuelles

le système est dit *uniforme*. Et si seulement un processus est particulier et exécute un protocole différent des autres, le système est dit *semi-uniforme*. Le site distingué est appelé *racine* ou *leader*. La deuxième hypothèse concerne la connaissance que peut avoir chaque sommet sur le réseau : sa topologie, le nombre de sommets, les sommets à certaine distance, etc.

1.1.2 Problèmes classiques

La conception d'applications distribuées dans un système distribué nécessite l'utilisation d'algorithmes de contrôle distribués. Ces algorithmes forment des primitives de base pour l'écriture d'une application distribuée. Nous pouvons citer entre autres :

- Élection : il consiste, à partir d'une configuration où tous les sites sont *candidats*, à atteindre une configuration où un site est déclaré *leader* et tous les autres sont déclarés *battus*,
- Exclusion mutuelle : de multiple ordinateurs partagent une ressource commune et unique dont l'accès ne peut se faire que par un seul processus à la fois. L'exclusion mutuelle doit aussi garantir l'accès à cette ressource par tous les processus,
- Synchronisation : la conception d'algorithmes sur systèmes synchrones est souvent plus facile que sur systèmes asynchrones. Tous les processus exécutent une action en même temps et ils sont tous commandés par une pulsation commune, générée par le système. Sur les systèmes asynchrones, chaque processus fonctionne à sa propre vitesse. L'ordre d'exécution des actions successives ne peut donc être déterminé à l'avance. Un algorithme conçu pour un système synchrone ne peut donc pas fonctionner directement sur

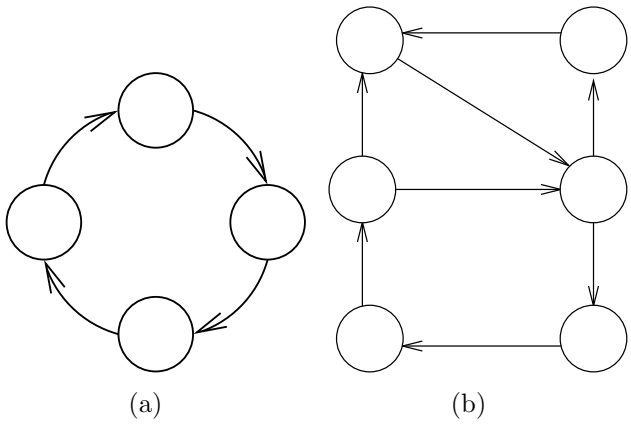


FIGURE 1.2 – Quelques topologies orientées usuelles

un système asynchrone.

Définition 1.1 *Un algorithme à vaques est un algorithme distribué qui satisfait les critères suivants :*

- *Terminaison : toute exécution est finie.*
- *Décision : toute exécution contient au moins un événement de décision.*
- *Dépendance : dans toute exécution, chaque événement est causalement précédé par un autre événement sur chaque processus.*

Définition 1.2 *Une exécution d'un algorithme à vagues s'appelle une vague ou un parcours.*

Les algorithmes utilisant un algorithme à vagues sous-jacent lancent plusieurs vagues. Pour cela, une topologie souvent particulière est utilisée, comme un anneau ou un arbre. Un algorithme qui exécute une succession de vagues est dit *perpétuel*. Ces algorithmes à vagues sont différenciés par les caractéristiques suivantes :

- Centralisation : un algorithme est dit *centralisé* si son exécution ne peut être déclenchée que par un et un seul initiateur.
- Topologie : certains algorithmes à vagues ne fonctionnent que sur des topologies particulières (anneau, clique, arbre, etc.),
- Connaissance pré-requise : l'algorithme peut nécessiter soit que tous les processus connaissent leur propre identité, celle de leurs voisins, l'orientation éventuelle du graphe, etc,
- Nombre de décisions : certains algorithmes à vagues permettent que plusieurs processus décident, d'autres ne permettent qu'à un seul de décider,
- Complexité : les différents algorithmes à vagues peuvent se différentier par leur complexité qui peut être calculée en nombre de messages échangés, en temps de calcul, en encombrement mémoire,
- Degré de parallélisation : il s'agit de la mesure du nombre de processus pouvant effectuer une action simultanément. Certains algorithmes ont un *degré de parallélisation* de degré 1. Dans ce cas, ils sont dit séquentiels.

Ces définitions sont très couramment employées dans les ouvrages [HR88, Ray87] et le lecteur intéressé pourra consulté ces références.

1.2 Tolérance aux fautes

Dans la section précédente, nous avons défini les systèmes distribués, dans lequel un ensemble d'entités (nœud, processus, processeur, ...) collaborent afin d'atteindre une tâche. Il est primordial que ces algorithmes tolèrent un maximum de fautes pouvant subvenir. Ces fautes sont des défaillances temporaires ou définitives d'un ou plusieurs composants du réseau. Un composant est dit défaillant lorsqu'il ne répond plus ses fonctions. Le but des algorithmes tolérants aux fautes est de garantir le bon fonctionnement du système malgré les dysfonctionnements des différents composants du système. Toutes ces fautes sont généralement classées suivant certains critères :

- l'origine de la faute : le type de composant qui est responsable de la faute, lien de communication ou processus,
- la cause de la faute : soit par omission, soit byzantine. Les fautes par omission regroupent les défaillances des composants alors que les fautes byzantines regroupent les comportements anormaux de ceux-ci tels qu'ils ne répondent plus aux spécifications qui les définissent,
- la durée de la faute : si la durée est supérieure au temps d'exécution de l'algorithme, elle est dite définitive sinon elle est dite transitoire ou intermittente,
- la détectabilité de la faute : une faute est détectable si le résultat de son exécution sur l'état d'un processus permet à celui-ci de détecter localement la faute.

Pour concevoir des algorithmes qui gèrent ces différentes défaillances, deux approches ont été proposées : la tolérance aux pannes qui garantit le respect des spécifications du problème malgré l'occurrence de fautes et l'auto-stabilisation qui garantit qu'un système, après l'occurrence d'une faute, retrouvera son comportement normal en un temps fini.
Dans la littérature, toutes les exécutions d'un algorithme tolérant aux fautes peuvent être classés suivant deux propriétés dites de *sûreté* et de *vivacité*.

Propriété de sûreté : elle assure qu'une propriété non voulue n'arrive jamais. Autrement dit, aucun problème n'est provoqué durant cette exécution.

Propriété de vivacité : elle assure qu'une propriété voulue finira par arriver.
Par exemple, si une exécution résout le problème du consensus binaire, les propriétés de sûreté et de vivacité peuvent s'énoncer de la façon suivante :

- sûreté : tous les processus se mettent d'accord sur une même valeur présente initialement dans le système. Si tous les processus corrects ont la même valeur initiale alors cette valeur est celle choisie pour le consensus,
- vivacité : le consensus entre les processus est obtenu en un temps fini.

Nous présentons maintenant trois approches de la tolérance aux fautes : la tolérance aux pannes, l'auto-stabilisation et la stabilisation instantanée.

1.2.1 Tolérance aux pannes

Lorsque certains composants sont en pannes, les algorithmes tolérants aux pannes essaient de maintenir un fonctionnement correct afin de continuer à vérifier les spécifications du système. Autrement dit les propriétés de sûreté et de vivacité doivent toujours être vérifiées. Mais avec

ces contraintes, tous les problèmes ne trouvent pas forcément de solution.
En 1985, Fisher, Lunch et Paterson ont montré dans [FLP85] qu'il est impossible de résoudre le problème du consensus défini ci-dessus, de manière déterministe dans un système asynchrone même si la défaillance est définitive et limitée à un seul processus. Cette impossibilité est obtenue sous la condition que la vivacité et la sûreté soient toujours vérifiées, même en présence de pannes.

1.2.2 Auto-stabilisation

L'auto-stabilisation a été introduite par Dijkstra en 1974 dans [Dij74] comme étant un système qui, quel que soit sa configuration initiale, est garanti d'arriver à une configuration légitime en un nombre fini d'étapes.

L'auto-stabilisation a été développée pour permettre aux systèmes distribués de gérer les fautes transitoires : telle la corruption des données locales et des messages en transit sur le réseau. Le principe de l'auto-stabilisation est qu'un système, après avoir subit une défaillance transitoire, finira par adopter un comportement qui respecte les spécifications du problème.

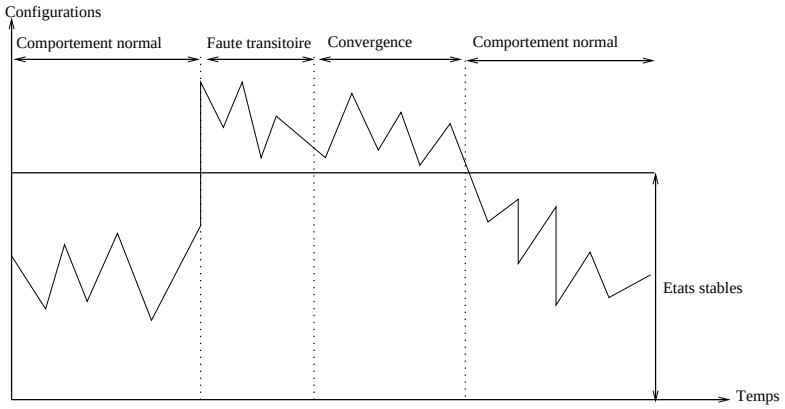


FIGURE 1.3 – Principe de l'auto-stabilisation

Les systèmes auto-stabilisants sont définis autours de deux propriétés : la *convergence* et la *clôture* du système.

Propriété de convergence : elle garantit qu'à partir d'une configuration quelconque le système retrouvera un comportement normal en un temps fini.

Propriété de clôture : elle assure que partant d'une configuration légitime et sans l'occurrence d'une faute, le système restera dans une configuration légale.
Ces deux propriétés garantissent que le système retrouvera un comportement normal à partir d'une configuration quelconque. La figure 1.2.2 illustre le principe de l'auto-stabilisation.

Stabilisation instantanée

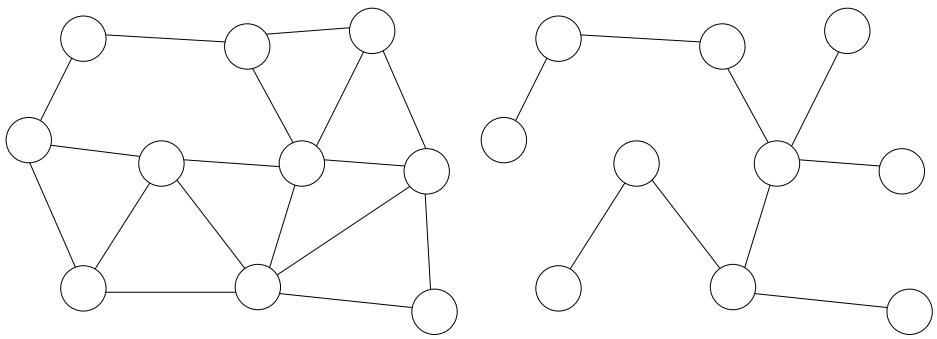
La stabilisation instantanée a été introduite par Bui, Datta, Petit et Villian dans [BDPV99]. Contrairement à l'auto-stabilisation, la stabilisation instantanée assure la sûreté au système

quelle que soit la configuration initiale. Un système instantanément stabilisant est un système qui se stabilise en 0 étape, c'est à dire quelle que soit sa configuration, il vérifie toujours les spécifications. Ainsi contrairement aux systèmes auto-stabilisants, les configurations "anormales" n'empêchent pas le système de vérifier les spécifications. Cette approche permet ainsi de limiter l'absence de sûreté à la période (à priori incontrôlable) correspondant à l'occurrence des défaillances transitoires.

1.2.3 Arbre couvrant

Avec le développement des réseaux informatiques et l'omniprésence des réseaux dans notre quotidien, les besoins de communiquer plus efficacement sont toujours grandissants. Ces réseaux offrent une multitude de services (partage de fichier, vidéo, ...) qui nécessitent l'échange d'information entre les nœuds constituant le réseau. Ces échanges doivent être optimisés afin de ne pas surcharger le réseau. La question cruciale est "comment faire circuler l'information sur le réseau".
L'approche la plus simple pour la communication serait d'utiliser le protocole d'inondation (*blind flooding*). Son fonctionnement est simple, chaque nœud qui reçoit un message de diffusion pour la première fois le transmet à tous ses voisins directs. Ainsi au bout d'un temps tous les nœuds du réseaux auront reçu le message. Malgré sa simplicité, cette méthode de communication entraîne une surcharge du réseau. De plus, chaque nœud reçoit plusieurs fois la même information, il y a donc une redondance inutile. Si on souhaite minimiser le nombre de messages envoyés et éviter la redondance alors la structure d'arbre répond à ces critères.

Un arbre couvrant $T = (V_T, E_T)$ du graphe $G = (V, E)$ est un graphe composant le même ensemble de nœuds $V_T = V$, mais seulement un sous ensemble $E_T \subseteq E$ d'arêtes tel qu'il existe exactement un unique chemin entre chaque paire de nœuds du réseaux [Gar03]. Cela signifie que le graphe est connexe (il y a au moins un chemin entre chaque deux nœuds) et il ne contient pas de cycles (il y a au plus un chemin entre chaque deux nœuds).



(a) Graphe d'un réseau (b) Arbre couvrant

FIGURE 1.4 – Exemple d'arbre couvrant

Pour un réseau donné il n'existe pas un unique arbre couvrant mais il peut y avoir une multitude.

1.3 Conclusion

Dans ce chapitre, nous avons défini de manière générale un système distribué. Ensuite, nous avons présenté les trois approches de la tolérance aux faute. Malheureusement, des défaillances peuvent survenir dans un réseau. Nous avons donc introduit les différents types de défaillances qui peuvent se produire et perturber le bon fonctionnement d'un système distribué. Pour tolérer les défaillances transitoires, l'approche d'auto-stabilisation permet de rendre un système distribué tolérant aux fautes. Cette approche est utilisée tout au long de cette thèse pour répondre les principaux problèmes classiques de l'algorithmique distribué. Les solutions proposées dans les chapitres suivants sont auto-stabilisantes.

Chapitre 2

Présentation des environnements mobiles

Résumé : *Nous présentons dans ce chapitre, les environnements mobiles. Un environnement mobile est un système constitué des unités mobiles et qui permet à ses utilisateurs d'accéder à l'information indépendamment de leurs positions géographiques. Nous décrivons les deux catégories des réseaux mobiles : réseaux avec infrastructure et réseaux ad hoc. Nous mettons ensuite en évidence les caractéristiques des réseaux ad hoc afin d'identifier les défis posés par ces réseaux. La mobilité des nœuds entraîne par exemple la réorganisation du réseau. L'absence d'infrastructure oblige les nœuds à agir à la fois comme terminaux pour communiquer avec les usagers et comme routeurs pour relayer le trafic pour le compte d'autres utilisateurs.* Pour plus de détails le lecteur intéressé pourra consulté l'ouvrage [MB07].

2.1 Introduction

Les communications sans fil ont un rôle crucial à jouer au sein des réseaux informatiques. Elles offrent des solutions ouvertes pour fournir de la mobilité ainsi que des services essentiels là où l'installation d'infrastructures n'est pas possible. Ces réseaux sont en plein développement du fait de leur flexibilité de leur interface, qui offre à un utilisateur la mobilité. Les environnements mobiles permettent une grande flexibilité d'emploi. En particulier, ils permettent la mise en place des réseaux dans des sites dont le câblage serait trop onéreux à réaliser dans leur totalité, voire même impossible (par exemple en présence d'une composante mobile). Ils sont aussi utilisés pour gérer la mise en place de secours lorsqu'une catastrophe (incendie, inondation, tremblement de terre, etc) a détruit les réseaux filaires.

Les réseaux mobiles sans fil sont classés en deux catégories : les réseaux avec infrastructure qui utilisent un nœud central, et les réseaux sans infrastructure ou les réseaux ad hoc. Un réseau mobile ad-hoc, consiste donc en un grand nombre d'unités mobiles se déplaçant dans un environnement quelconque en utilisant, comme moyen de communication, des interfaces sans fils.

Bien que l'environnement mobile offre beaucoup d'avantages par rapport à l'environnement habituel, de nouveaux problèmes propres à l'environnement mobile peuvent apparaître : une fréquente déconnexion, un débit de communication modeste, des sources d'énergie limitées, etc.

Dans ce chapitre nous présentons les environnements mobiles et les principaux concepts liés à ces environnements. Nous commençons par définir cet environnement et citer les deux classes qui le constituent. Nous introduisons ensuite le concept des réseaux ad hoc et les caractéristiques inhérentes à ces réseaux.

2.1.1 Les environnements mobiles

Un environnement mobile est un système composé de sites mobiles et qui permet à ses utilisateurs d'accéder à l'information indépendamment de leurs positions géographiques. Les réseaux mobiles ou sans fil, peuvent être classés en deux catégories : les réseaux avec infrastructure et les réseaux sans infrastructure.

Les réseaux avec infrastructure

Le modèle de réseau mobile avec infrastructure intègre deux ensembles d'entités distinctes : les sites fixes d'un réseau de communication filaire classique (*wired network*), et les sites mobiles (*wireless network*). Certains sites fixes, appelés stations support mobile *(Mobile Support Station)* ou station de base (SB) sont munis d'une interface de communication sans fil pour la communication directe avec les sites ou unités mobiles (UM), localisés dans une zone géographique limitée, appelée cellule (voir figure 2.1).

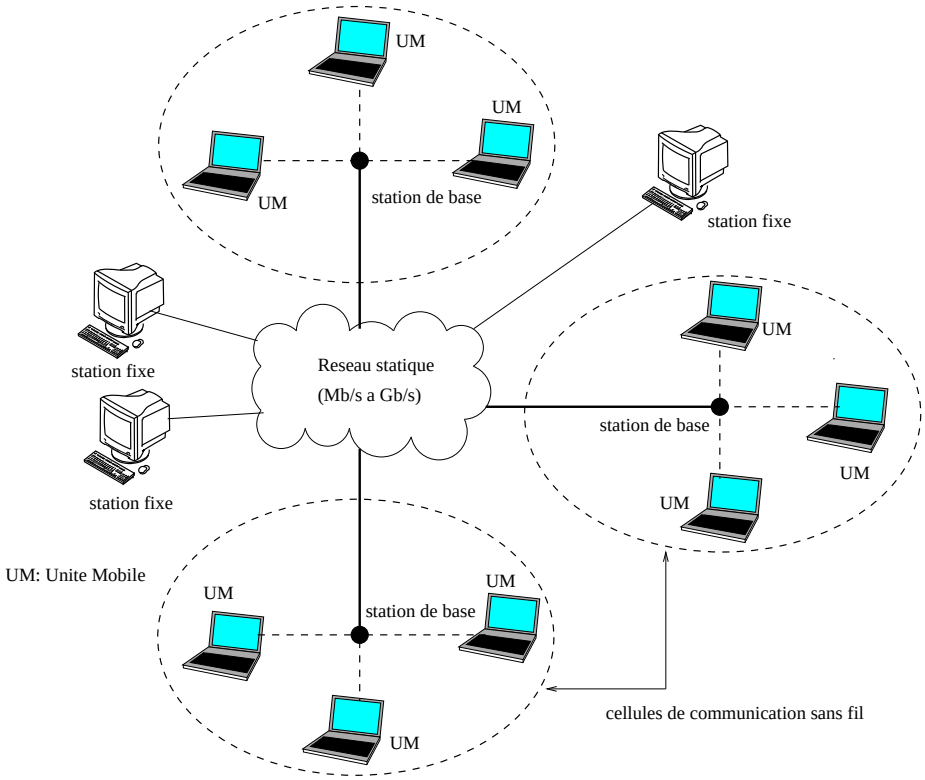


FIGURE 2.1 – Le modèle des réseaux mobiles avec infrastructure

A chaque station de base correspond une cellule à partir de laquelle des unités mobiles peu-

vent émettre et recevoir des messages. Les sites fixes sont interconnectés entre eux à travers un réseau de communication filaire, généralement fiable et d'un débit élevé. Les liaisons sans fil ont une bande passante limitée qui réduit sévèrement le volume des informations échangées [DD92]. Dans ce modèle, une unité mobile ne peut être, à un instant donné, directement connectée qu'à une seule station de base. Elle peut communiquer avec les autres sites à travers la station à laquelle elle est directement rattachée.

Les réseaux sans infrastructure

Le modèle de réseau mobile sans infrastructure préexistante ne comporte pas de site fixe. Tous les sites du réseau sont mobiles et communiquent d'une manière directe en utilisant leurs interfaces de communication sans fil (voir figure 2.2). L'absence d'infrastructure ou de réseau filaire composé de stations de base, oblige les unités mobiles à se comporter comme des routeurs qui participent à la découverte et la maintenance des chemins pour les autres hôtes du réseau.

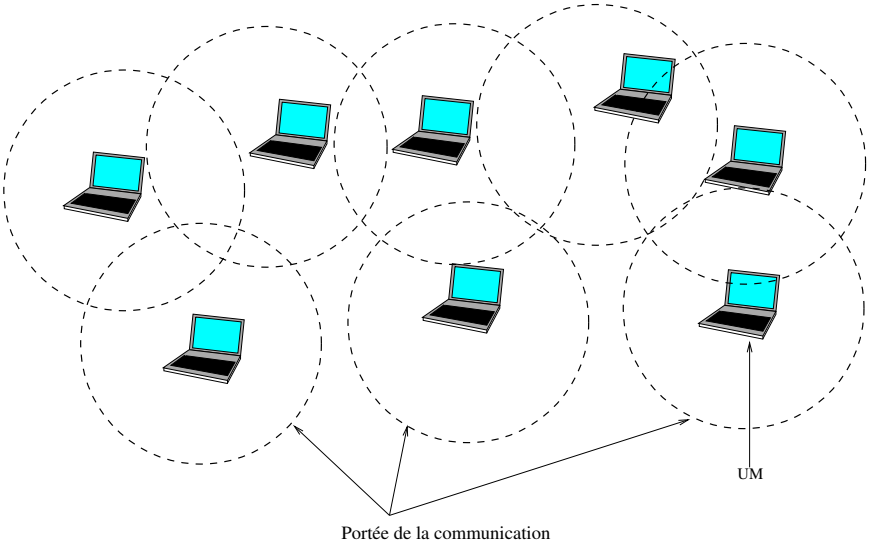


FIGURE 2.2 – Le modèle des réseaux mobiles sans infrastructure

2.2 Les réseaux ad hoc

Un réseau mobile ad hoc appelé généralement MANET (Mobile Ad hoc Network) est un réseau composé des terminaux mobiles qui communiquent sans nécessiter d'infrastructure fixe préexistante. Les réseaux ad hoc représentent donc une alternative intéressante aux réseaux à infrastructure, notamment pour les applications distribuées qui sont dynamiquement déployées, comme par exemple les applications dans le domaine militaire.

Les systèmes de communication cellulaire sont basés essentiellement sur l'utilisation des réseaux filaires et la présence des stations de base qui couvrent les différentes unités mobiles du système. Ces réseaux (réseaux cellulaires) requièrent également un important effort de planification pour leur déploiement. Les réseaux mobiles ad hoc sont à l'inverse, des réseaux capables

de s'organiser sans infrastructure préalablement prédéfinie en permettant des échanges directs entre stations mobiles. Ainsi, le fonctionnement du réseau repose sur les stations elles-mêmes : celles-ci interviennent à la fois comme terminaux pour communiquer avec les usagers et comme routeurs afin de relayer le trafic pour le compte d'autres utilisateurs. En particulier, lorsqu'un émetteur ne peut communiquer directement avec le destinataire parce qu'il n'est pas à portée directe de la machine destination, les informations devront être transmises de proche en proche avant d'atteindre la destination souhaitée.
Les réseaux ad hoc permettent d'offrir de la connectivité avec un déploiement rapide et à faible coût. De plus, aucune contrainte ne limite sa taille en termes d'étendue ou de nombre d'unités.

Les réseaux ad hoc sont des réseaux fortement dynamiques. Ainsi, les nœuds du réseaux sont libres de se déplacer et de s'organiser arbitrairement, impliquant une grande variabilité de la topologie du réseau comme le montre la figure 2.3.

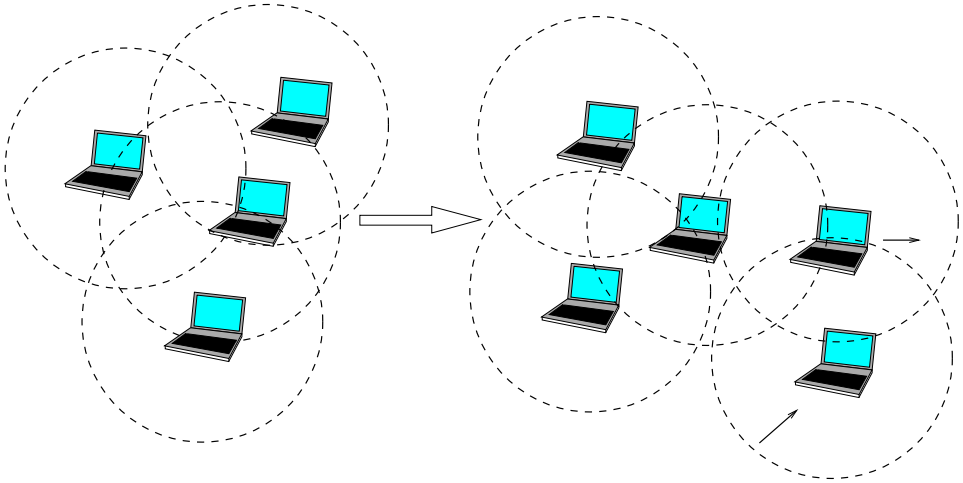


FIGURE 2.3 – Changement de topologie d'un réseau ad hoc

Nous remarquons bien que l'arrivée et le départ des nœuds sont très fréquents.

2.2.1 Applications des réseaux ad hoc

La particularité du réseau ad hoc est qu'il n'a besoin d'aucune installation fixe, ceci lui permettant d'être rapide et facile à déployer. Les opérations tactiques comme les opérations de secours, militaires ou d'explorations trouvent en ad hoc, le réseau idéal. La technologie ad hoc intéresse également la recherche des applications civiles. On distingue entre autre :

- Les services d'urgence : opération de recherche et de secours des personnes, tremblement de terre, feux, inondation, dans le but de remplacer l'infrastructure filaire,
- Le travail collaboratif et les communications dans des entreprises ou bâtiments : dans le cadre d'une réunion ou d'une conférence par exemple,
- Applications commerciales : pour un paiement électronique distant (taxi) ou pour l'accès mobile à l'Internet, où service de guide en fonction de la position de l'utilisateur,
- Réseaux de senseurs : pour des applications environnementales (climat, activité de la terre, suivi des mouvements des animaux, etc.) ou domestiques (contrôle des équipements

à distance).

Les applications potentielles des réseaux ad hoc sont nombreuses. Par exemple, on peut penser qu'un groupe de personnes avec des ordinateurs portables lors d'une conférence qui souhaite échanger des fichiers peut rapidement mettre en place un réseau ad hoc sans avoir recours à une infrastructure supplémentaire. Les réseaux ad hoc sont idéals dans des zones où un tremblement de terre ou d'autres catastrophes naturelles ont détruit les infrastructures de communication.
D'une façon générale, les réseaux ad hoc sont utilisés dans toute application où le déploiement d'une infrastructure réseau filaire est trop contraignant, soit parce que difficile à mettre en place, soit parce que la durée d'installation du réseau ne justifie pas de câblage à demeure.

2.2.2 Caractéristiques

La communication de données dans un réseau ad hoc diffère de celle des réseaux câblés dans différents aspects. Le moyen de communication sans fil n'a pas un comportement prévisible comme dans un canal câblé. Au contraire, le moyen de communication sans fil a des caractéristiques variables et imprévisibles. Contrairement à un réseau câblé, le médium sans fil est un médium de diffusion, c'est-à dire, tous les nœuds qui sont dans la portée de transmission de l'émetteur peuvent recevoir le message. Par contre, ces réseaux permettent de réduire considérablement le coût de déploiement.
De manière générale, l'utilisation effective des réseaux ad hoc est rendue difficile par les spécificités de ces réseaux qui sont entre autres :

- La topologie est dynamique : les réseaux ad hoc sont formés spontanément à partir des nœuds mobiles sans nécessiter une infrastructure fixe. Ces nœuds peuvent se déplacer de façon libre et arbitraire. Ainsi, un nœud peut quitter ou rejoindre le réseau à tout instant. Par conséquent, la topologie du réseau peut changer à des instants imprévisibles, d'une manière rapide et aléatoire. Cela affecte fortement la disponibilité des chemins de routage, et les protocoles de routage doivent s'adapter à la mobilité des nœuds,
- la bande passante est limitée : une des caractéristiques primordiales des réseaux basés sur la communication sans fil est l'utilisation d'un médium de communication partagé. Ce partage fait que la bande passante réservée à un hôte soit modeste,
- les contraintes énergétiques sont fortes : chaque unité doit bien souvent embarquer une alimentation autonome. Dans les réseaux ad hoc les nœuds agissent comme routeurs afin de relayer le trafic pour le compte d'autres utilisateurs, ce qui fait qu'une partie de l'énergie est déjà consommée par la fonctionnalité du routage,
- L'absence d'infrastructure : les réseaux ad hoc se distinguent des autres réseaux mobiles par la propriété d'absence d'infrastructure préexistante et de tout genre d'administration centralisée. Les hôtes mobiles sont responsables d'établir et de maintenir la connectivité du réseau d'une manière continue, ainsi chaque nœud participe à la survie du réseau,
- Une sécurité physique limitée : les réseaux ad hoc sont plus touchés par le paramètre de sécurité, que les réseaux filaires classiques. Pour les réseaux ad hoc, le principal problème ne se situe pas tant au niveau du support physique mais principalement dans le fait que tous les nœuds sont équivalents et potentiellement nécessaires au fonctionnement du réseau,
- L'hétérogénéité des nœuds : les nœuds ad hoc peuvent correspondre à une multitude

d'équipements, par exemple des ordinateurs portables, des PDA, téléphones mobiles, etc. Ainsi, ces nœuds peuvent avoir des différences en termes de capacité de traitement (CPU, mémoire), de mobilité (lent, rapide) et de logiciel, mais ils doivent inter-opérer pour maintenir le réseau.

Compte tenu de toutes ces différences, la conception d'algorithmes pour les réseaux ad hoc sont plus complexes que leurs homologues câblés.

2.3 Conclusion

Dans ce chapitre, nous avons présenté les concepts des environnements mobiles et en particulier les réseaux ad hoc. Ces environnements sont caractérisés par de fréquentes déconnexions des nœuds et des restrictions sur les ressources utilisées, surtout si tous les usagers du système sont mobiles ce qui est le cas pour les réseaux ad hoc. Ces limitations transforment certains problèmes ayant des solutions évidentes dans l'environnement classique, en des problèmes complexes et difficiles à résoudre. Cette présentation des réseaux ad-hoc permet aussi de les situer dans l'évolution des réseaux sans fil et de comprendre les difficultés du routage des messages dans de tels réseaux.
Bien que ces réseaux présentent des avantages énormes, malheureusement beaucoup de problèmes restent à résoudre, notamment le problème du routage des messages. Ainsi, toute conception des protocoles pour les réseaux ad hoc doit prendre en compte les spécificités de ces réseaux.

Chapitre 3

Algorithmes de clustering

Résumé : *Dans ce chapitre, nous présentons d'abord les différentes approches utilisées pour structurer un réseau en clusters. Nous décrivons les deux catégories des algorithmes de clustering : les algorithmes de clustering à 1 saut et les algorithmes de clustering à k sauts. Cette étude relève les limites des solutions existantes et nous sert de support pour positionner nos travaux de recherche. Nous présentons ensuite notre solution pour la structuration du réseau en clusters. L'algorithme que nous proposons est un algorithme auto-stabilisant qui est basé sur des connaissances locales. Avec notre solution, l'ensemble des phases de la découverte de topologie et la structuration du réseau ne nécessite qu'un unique message. Cette approche consiste à combiner les deux phases en une seule et à fabrique des clusters disjoints. Nous avons montré formellement que dans le pire des cas (la chaîne linéaire ordonnée), nous avons un temps de stabilisation de n+2 transitions, avec n le nombre de nœuds du réseau. Nous avons effectué un ensemble de simulations sous différents scénarios et les résultats obtenus montrent la pertinence de l'organisation proposée. À partir de cette solution, nous proposons deux applications : une diffusion d'informations dans le réseau et un protocole de routage que nous présentons dans les chapitres suivants.*

3.1 Introduction

Un réseau ad-hoc sans fil est composé essentiellement d'hôtes mobiles qui communiquent les uns avec les autres sans infrastructure fixe et sans administration centrale. Les unités mobiles du réseau, se déplacent d'une façon libre et arbitraire. Par conséquent, la topologie du réseau peut changer, à des instants imprévisibles, d'une manière rapide et aléatoire. La communication de données dans le réseau ad hoc diffère de celle des réseaux classiques. Le moyen de communication sans fil n'a pas un comportement prévisible comme dans les réseaux classiques. En revanche, le moyen de communication sans fil a des caractéristiques variables et imprévisibles, par exemple une bande passante limitée, des sources d'énergie limitée etc...
Dans ce contexte, un des problèmes majeurs est l'établissement et la maintenance des routes entre les nœuds du réseau. Un réseau ad hoc est classiquement considéré comme un réseau non structuré c'est à dire tous les nœuds du réseau ont des rôles égalitaires . Avec un tel réseau, lorsque la taille du réseau grandit le nombre de messages (messages de contrôle et routage) dans le réseau augmente. Cela se traduit par une forte dégradation de performance du réseau. Partant de ce constat, nous pensons qu'un réseau ad hoc doit être organisé pour faciliter son

utilisation. Ainsi la structuration est une approche importante pour simplifier le fonctionnement d'un réseau ad hoc. Pour structurer un réseau ad hoc, plusieurs approches ont été proposées dans la littérature. Ces approches se basent essentiellement sur la technique de clustering. Le clustering vise principalement à optimiser les informations échangées pour la maintenance de topologie et réduire *l'overhead* du réseau en évitant les *broadcasts*.

Dans ce chapitre, nous allons dans un premier temps définir le clustering. Ensuite, nous présenterons dans la section 3.1.2 les objectifs de clustering, puis dans la section 3.1.3 nous décrirons les propriétés fondamentales que doivent avoir une structure en clusters. La section 3.2 présentera un état de l'art des algorithmes de clustering dans les réseaux ad hoc. La section 3.3 décrira notre solution.

3.1.1 Définition

Le clustering consiste à découper le réseau en groupes d'entités appelés clusters en donnant au réseau une structure hiérarchique [JN06], [Mit06]. Chaque cluster est représenté par un nœud particulier appelé clusterhead (voir figure 3.1.1). Ce nœud est élu comme clusterhead selon une métrique spécifique ou une combinaison de métriques telles que l'identifiant, le degré, la mobilité, le poids, la densité...
Le clusterhead agit comme un coordinateur local dans son cluster. Un cluster est donc composé d'un clusterhead, de nœuds de passage et éventuellement de nœuds appelés ordinaires.

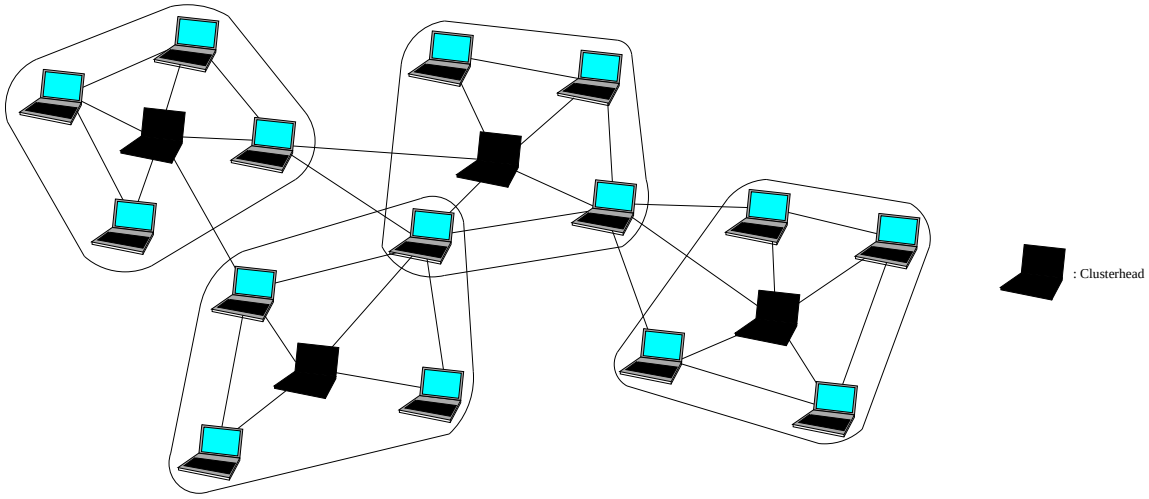


FIGURE 3.1 – Structure en clusters

3.1.2 Avantages du clustering

Comme nous l'avons vu, le principe du clustering consiste à organiser le réseau en une structure hiérarchique. Cette structure hiérarchique permet de :

– Optimiser la bande passante en minimisant la quantité d'information échangée afin de maintenir les tables de routage,

- Rendre le routage hiérarchique. L'idée du routage hiérarchique est d'établir des schémas de routage différents à l'intérieur des clusters et entre les clusters. Ainsi chaque nœud du réseau stocke la totalité des informations de son cluster et une partie des informations concernant les autres clusters, ce qui minimise considérablement la taille des tables de routage et le nombre de messages échangés dans le réseau et rend le routage plus efficace,
- Redistribuer les ressources à travers le réseau et contrôler la ré-utilisation spatiale des fréquences [Bas99],
- Optimiser la diffusion d'information afin de ne pas dégrader les performances du réseau. L'idée est de permettre à certains nœuds de relayer l'information afin qu'elle soit diffusée dans tout le réseau,
- Faciliter la ré-utilisation des ressources, cela permet d'améliorer la capacité du système [JN06].

3.1.3 Propriétés fondamentales d'une structure en clusters

Comme nous avons pu le constater, une structure en clusters présente de nombreux avantages. Il est donc nécessaire que cette structure présente les propriétés que nous décrirons dans cette partie.
Les propriétés que doivent posséder une structure en clusters sont :

- Minimiser les informations du réseau : pour concevoir une telle structure, il est nécessaire d'utiliser un algorithme basé sur des connaissances locales. De tels algorithmes sont appelés des algorithmes localisés [GCSRS08]. Un algorithme localisé est un algorithme distribué qui permet à un nœud du réseau de communiquer en échangeant des messages qu'avec des nœuds à une distance bornée de lui. Donc chaque nœud du réseau prend une décision en fonction des informations locales qu'il possède. D'où chaque nœud doit avoir une vue partielle du réseau, moins de trafic de contrôle est requis et réduit ainsi le gaspillage de bande passante,
- Adaptation à l'environnement : puisque chaque nœud du réseau se déplace d'une façon libre et arbitraire, impliquant une grande variabilité de la topologie du réseau. En conséquence, l'algorithme doit s'adapter aux changements topologiques. En cas de modification topologique (disparition d'un nœud, apparition d'un nœud, etc) le système doit s'adapter et se reconstruit rapidement sans aide extérieure,
- Passage à l'échelle : l'algorithme doit supporter le passage à l'échelle. Même si le nombre de nœuds augmente, les performances du réseau ne doivent pas diminuer. Donc aucune congestion due à un nombre important de nœuds dans le réseau ne doit se produire. Un algorithme est dit efficace si les performances du réseau ne doivent pas chuter d'une manière drastique quand le nombre de nœuds augmente dans le réseau.

Il est également important de souligner que la performance d'un réseau ad hoc dépend aussi de l'interaction entre les nœuds du réseau. Aucun nœud à lui seul ne peut construire ou maintenir une structure mais chaque nœud doit contribuer à la construction et à la maintenance du système. Avant de construire une structure, les nœuds découvrent d'abord la topologie du réseau en envoyant des messages de type *hello*. Ensuite ils utilisent cette information pour construire la structure. Si par exemple un nœud peu coopérant refuse de répondre, cela peut conduire à des situations d'incohérences dans la prise de décision des nœuds. Ainsi la coopération est une propriété importante dans les réseaux ad hoc.

3.2 Les algorithmes de clustering des réseaux ad hoc

Dans cette section, nous présenterons les principaux algorithmes de clustering dans les réseaux ad hoc. Dans la littérature, il existe plusieurs solutions pour organiser un réseau en clusters. Toutes ces solutions sont destinées à identifier un sous ensemble de nœuds du réseau (appelés clusters). Les clusters sont identifiés par leur clusterhead. Les différents algorithmes se distinguent sur le critère de sélection des clusterheads, c'est à dire la métrique. Cette métrique peut être une métrique spécifique comme l'identifiant, le degré, la mobilité, l'énergie, ... ou une combinaison de métriques. Il existe deux grandes catégories d'algorithmes de clustering : les algorithmes de clustering à 1 saut et les algorithmes de clustering à k sauts. Les algorithmes de clustering à 1 saut construisent des clusters où chaque nœud du réseau est à distance 1 de son clusterhead. Par contre les algorithmes de clustering à k sauts construisent des clusters où chaque nœud est à distance k de son clusterhead. Nous allons maintenant présenter les principaux algorithmes existants dans la littérature.

3.2.1 Approches générant des clusters à 1 saut

Il existe de nombreux algorithmes de clustering à 1 saut. Ephremides, Weiselthier et Baker ont proposé dans [EWB88] l'un des premiers algorithmes de clustering pour les réseaux ad hoc. Il s'agit de l'algorithme de plus petit ID appelé aussi Linked Cluster Architecture (LCA). Chaque nœud du réseau doit avoir un identifiant unique appelé ID et chaque nœud se déclare clusterhead ou non en se basant sur son identifiant et ceux de ses voisins. A la fin du processus de formation, chaque nœud du réseau doit avoir l'un des statuts suivants : clusterhead, nœud membre ou nœud de passage. Au début tous les nœuds du réseau ont un statut de nœud membre. Dans cet algorithme, si un nœud u possède le plus petit identifiant parmi tous ses voisins à 1 saut, il se déclare clusterhead et ses voisins à 1 saut dont les identifiants sont supérieurs à celui du clusterhead u le joignent et deviennent des nœuds membres. Sinon u attendra jusqu'à ce que tous ses voisins à 1 saut ayant des identifiants plus petits que lui diffusent leur décision et que si ces nœuds ont tous un statut de nœud membre (dans ce cas ils se sont déjà attaché à un clusterhead voisin de plus petit ID) alors u se déclare clusterhead. Une fois que les nœuds ont soit un statut de nœud membre ou clusterhead, alors si un nœud se trouve être entouré par deux ou plusieurs clusterheads, il deviendra nœud de passage. Un cluster est formé par le clusterhead et tous ses membres. Cet algorithme est exécuté en mode synchrone tel qu'à chaque nœud est attribué un intervalle de temps fini appelé slot (TDMA) [1] (le but de ce principe est d'éviter que des collisions ne se produisent dans le réseau). L'algorithme LCA construit à la fois des clusters recouvrants (c'est à dire qu'il existe des nœuds du réseau qui appartiennent à 2 clusters simultanément) et non-recouvrants (voir figure 3.2.1).

Gerla et Tsai ont proposé dans [GT95] un algorithme de clustering appelé High-Connectivity Clustering(HCC). Cet algorithme est basé sur le degré de connectivité (nombre de voisins du nœud) pour construire des clusters au lieu des identités des nœuds. Un nœud est élu clusterhead, s'il a la plus haute connectivité parmi tous ses voisins à 1 saut. Si deux nœuds ont le même degré de connectivité, alors le nœud ayant le plus petit identifiant deviendra clusterhead. L'algorithme HCC génère un nombre réduit de clusters puisqu'il favorise les nœuds ayant le plus

1. Time Disivion Multiple Access

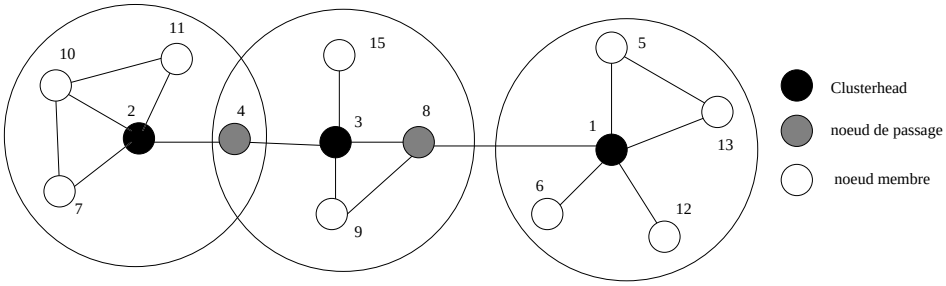


FIGURE 3.2 – Construction de clusters avec LCA

fort degré d'être clusterheads, c'est à dire les nœuds qui couvrent plus de nœuds voisins. Mais cet algorithme souffre de fréquents changements de clusterheads, en raison du critère choisi pour la sélection du clusterhead. Ainsi dans cet algorithme, les clusterheads ne sont pas susceptibles de garder leur statut très longtemps puisque leurs degrés changent fréquemment contrairement à l'algorithme LCA où les nœuds ont tendance à garder leur statut plus longtemps. Comme dans LCA, cet algorithme construit à la fois des clusters recouvrants et non-recouvrants.

Dans les algorithmes LCA et HCC, l'opération de maintenance des clusters est très coûteuse parce que le mouvement d'un nœud peut détruire la structure et nécessiter une reconstruction entière de la structure. C'est pourquoi les auteurs ont proposé dans [CWLG97] un algorithme appelé Least Cluster Change (LCC). Cet algorithme permet d'améliorer la maintenance des clusters construits avec l'algorithme LCA ou HCC et d'apporter plus de stabilité dans la composition des clusters. Le but principal de cet algorithme est de minimiser le coût de la reconstruction des clusters. La reconstruction des clusters survient seulement si les deux conditions suivantes sont vérifiées :

- si un nœud sort complètement dans la zone de communication de tous les clusterheads du réseau,
- si deux clusterheads deviennent voisins.

Dans cet algorithme, quand un nœud non clusterhead u se déplace d'un cluster C_i vers un cluster C_j alors la réélection du clusterhead dans le cluster C_j n'aura pas lieu, même si le nœud u possède la priorité pour être clusterhead dans le cluster C_j. Cependant, quand un nœud u non clusterhead sort de son cluster et ne rentre pas dans les clusters existants, le nœud u devient clusterhead. En outre quand deux clusterheads deviennent voisins, alors l'un des deux clusterheads doit abandonner son statut de clusterhead selon le critère utilisé pour construire les clusters (plus haute connectivité et/ou plus petite identité). De cette façon, l'algorithme LCC apporte une meilleure stabilité de la structure mais n'évite pas complétement la reconstruction des clusters vu qu'un seul nœud peut relancer le processus de clustering s'il n'existe pas de clusterhead dans son voisinage.

Besagni a proposé dans [Bas99] deux algorithmes de clustering utilisant une nouvelle approche. Le premier, Distributed Clustering Algorithm (DCA) qui est destiné aux réseaux "quasi-static" dans lequel les déplacements des nœuds doivent être "lents". Cet algorithme attribue un poids générique à chaque nœud du réseau. Le critère d'élection du clusterhead est le poids maximal dans le voisinage. Un nœud u est élu clusterhead, s'il possède le plus grand poids dans son voisinage à 1 saut. Ainsi, l'algorithme DCA semble bien adapté aux réseaux dans lesquels

les nœuds sont statiques ou se déplacent avec une vitesse relativement lente. Le second algorithme est destiné aux réseaux de grande mobilité et appelé Distributed and Mobility-Adaptive Clustering algorithm (DMAC). Comme dans DCA, DMAC associe également un poids à chaque nœud du réseau. Le critère utilisé pour le choix du clusterhead est le même que celui utilisé dans DCA. Chaque nœud réagit localement à toutes les variations en fonction de son statut : clusterhead ou nœud membre. Dans les 2 algorithmes, il est supposé que chaque nœud a la connaissance de son poids. Un nœud est choisi pour être clusterhead si son poids est plus grand que parmi tous ses voisins à 1 saut. Les deux techniques se basent sur l'algorithme LCA et changent juste le critère de choix du clusterhead. Le but de ces approches est d'apporter plus de stabilité en définissant le poids par les paramètres de la mobilité du nœud. Par exemple, quand le poids attribué aux nœuds est proportionnel à leur vitesse de déplacement, les nœuds moins mobiles seront choisis pour être clusterheads.

Dans [JN06], les auteurs ont proposé un algorithme de clustering pour les réseaux de capteurs. Cet algorithme utilise le poids associé aux nœuds comme critère dans le choix des clusterheads. Le poids peut être la bande passante, l'espace mémoire ou l'autonomie de batterie. Un nœud est choisi pour être clusterhead, s'il possède le plus grand poids parmi tous ses voisins à un 1 saut. Si deux clusterheads u et v deviennent voisins, alors le clusterhead u de plus petit poids abandonnera son statut de clusterhead.

Dans [YC03], les auteurs ont proposé un algorithme de clustering appelé 3-hop Between Adjacent Clusterheads (3hBAC). 3hBAC construit des clusters disjoints et impose 3 sauts entre deux clusterheads de clusters adjacents. Le critère du choix de clusterhead est le degré du nœud (nombre de voisins du nœud). Un nœud ayant le plus grand degré se déclare clusterhead, et ses voisins à 1 saut s'attachent à lui et se déclarent nœuds membres. Les nœuds voisins de ces nœuds membres et non voisins d'un clusterhead se déclarent "nœuds non spécifiés". Ces nœud ne peuvent pas être des clusterheads. 3hBAC utilise également l'algorithme LCC pour la maintenance de clusters. Si deux clusterheads u et v se retrouvent voisins, le clusterhead u de plus petit degré abandonne son statut de clusterhead et devient un nœud membre. Ses voisins deviennent soit nœuds membres s'ils ont un voisin clusterhead, soit "nœuds non spécifiés" (voir figure 3.3). Comme la distance entre deux clusterheads est de 3 sauts, il y a peu de chance que deux clusterheads deviennent voisins. Cette technique permet de réduire le nombre de clusters et évite la reconstruction entière de la structure en cas de modification topologique. 3hBAC fournit de bonnes performances que HCC en termes de nombre de clusters formés et le temps moyen qu'un clusterhead ou nœud membre garde son statut. Cependant, cet algorithme exige que chaque nœud maintient deux tables : une table de voisinage et une table de nœuds membres qui contient tous les membres du réseau. Cela peut être coûteuse en termes de mémoire et de messages échangés.

Dans [LG97], les auteurs présentent un algorithme appelé Adaptive Clustering for Mobile Wireless Networks qui construit de clusters non-recouvrants. Dans cet algorithme, une fois les clusters formés, la notion de clusterhead disparaît et tous les nœuds auront des rôles égalitaires. Le statut de clusterhead est juste utilisé pour construire les clusters. En donnant un rôle prépondérant aux clusterheads, cela peut créer un goulot d'étranglement dans le réseau. Par exemple dans le cas du routage, les clusterheads sont responsables de la découverte des

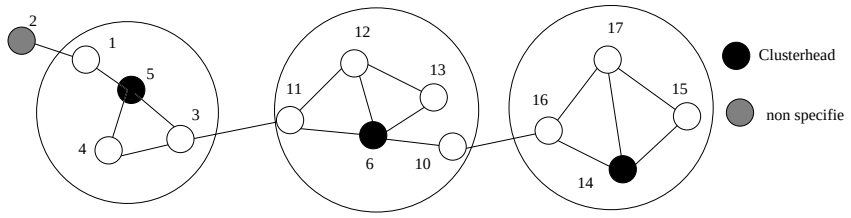


FIGURE 3.3 – Construction de clusters avec 3hBAC

routes dans le réseau. La motivation des auteurs est d'éviter ce genre des problèmes. Pour construire de tels clusters, les auteurs supposent que chaque nœud connaît tous ses voisins à 1 saut et que la topologie du réseau ne change pas durant l'exécution de l'algorithme. Chaque nœud maintient un ensemble N qui initialement contient les identifiants de tous ses voisins à 1 saut. Un nœud diffuse son statut (clusterhead, membre, non spécifié) que s'il possède un identifiant plus petit que les nœuds de N. Un nœud s'élit clusterhead que s'il a un identifiant plus petit que tous les identifiants de son ensemble N. Sur réception du statut d'un nœud u, les voisins de u suppriment u dans leur ensemble N. Si u est clusterhead, les voisins s'attachent à lui s'ils n'étaient pas encore membres d'aucun cluster ou le clusterhead auquel ils étaient attachés avait un identifiant plus grand que u. Le processus se répète jusqu'à ce que l'ensemble N de chaque nœud devient un ensemble vide. L'opération de maintenance des clusters nécessite la connaissance des voisins à deux sauts. Ainsi, chaque nœud doit connaître son voisinage à deux sauts. De cette façon, chaque nœud sait si les membres de son cluster restent toujours à deux sauts de lui. Si deux nœuds du même cluster se retrouvent éloignés de plus de deux sauts, seul celui encore voisin du nœud de plus grand degré dans le cluster reste dans le cluster. L'autre nœud doit s'attacher à un autre cluster. Si un nœud change de cluster, il doit d'abord vérifier que les membres du cluster sont tous à deux sauts de lui. Bien que cet algorithme ne surcharge pas les clusterheads, il nécessite la connaissance des voisins à deux sauts dans la maintenance de clusters. Par conséquent, il génère un nombre important de messages de contrôle.

Dans le but d'apporter plus de stabilité de la structure, les auteurs ont proposé dans [BKL01] un algorithme basé sur la mobilité appelé Lowest Relative Mobility Clustering Algorithm (MOBIC). Cet algorithme se base également sur l'algorithme LCA et utilise la mobilité relative des nœuds comme critère dans le choix du clusterhead au lieu d'utiliser le degré. L'idée est de permettre aux nœuds moins mobiles de jouer le rôle de clusterhead car ils apportent plus de stabilité. Pour calculer la mobilité relative, chaque nœud du réseau mesure le niveau du signal qui l'unit à chacun de ses voisins. A l'étape suivante, chaque nœud u calcule la mobilité en faisant le rapport entre son niveau de puissance et celui mesuré à l'étape précédente pour chaque voisin de u, le quotient du rapport obtenu représente la mobilité. Si le niveau de signal mesuré entre deux nœuds u et v est supérieur à celui mesuré à l'étape précédente alors les deux nœuds se rapprochent de l'un de l'autre, dans le cas contraire ils s'éloignent de l'un de l'autre. En d'autre terme, plus la distance entre les nœuds est importante, plus le signal s'atténue. Un nœud est élu clusterhead, s'il possède la plus faible mobilité parmi tous ses voisins à 1 saut. En cas d'égalité, le nœud ayant la plus petite identité sera choisi pour être clusterhead. D'autre part, MOBIC utilise l'algorithme LCC pour la maintenance des clusters en ajoutant une règle supplémentaire : si deux clusterheads u et v deviennent voisins, alors le clusterhead u de plus

petit identifiant n'abandonnera son statut de clusterhead que si v reste toujours voisin de u après une certaine période t. Après cette période, si les deux nœuds restent toujours voisins alors u abandonnera son statut de clusterhead. Cela permet d'éviter la reconstruction de la structure si les deux clusterheads restent voisins que pour une courte période. Bien que la prise en compte de la mobilité des nœuds dans le choix du clusterhead semble intéressante, la méthode utilisée pour estimer la puissance du signal est un peu complexe.

Dans [BS01], les auteurs ont proposé un algorithme appelé Mobility-Based Clustering (MBC) basé également sur le concept de la mobilité relative pour construire des clusters. Pour calculer le mobilité relative, chaque nœud mesure le niveau du signal qui l'unit à chacun de ses voisins. Le quotient du rapport entre ce niveau de puissance et celui mesuré à l'étape précédente représente la mobilité relative. Cet algorithme nécessite l'utilisation d'un GPS [2] pour permettre à chaque nœud de localiser ses voisins. Un nœud u est élu clusterhead, si u possède la plus faible mobilité relative parmi tous ses voisins. Bien que cet algorithme apporte de la stabilité dans la reconstruction des clusters, il souffre de la réévaluation fréquente de la mobilité. Par exemple quand un nœud se déplace, sa mobilité doit également changer. Par conséquent la mobilité du nœud doit être réévalué périodiquement pour permettre aux nœuds de s'adapter aux changements topologiques.

Dans [LMHCH06], les auteurs ont proposé un algorithme appelé multicast power greedy clustering basé sur une heuristique afin de réduire la consommation d'énergie et ainsi prolonger la durée de vie du réseau. Dans cet algorithme, les auteurs supposent que chaque nœud peut contrôler son niveau de transmission du signal et chaque nœud doit avoir plusieurs niveaux de transmission. Cet algorithme s'exécute en trois phases consécutives : beacon phase, greedy phase et recruiting phase. Dans la première phase, chaque nœud envoie un signal beacon avec la plus grande puissance afin d'informer son voisinage de sa présence. Chaque nœud collectera les informations de ses voisins à la réception du signal beacon. Les auteurs supposent aussi qu'au cours de cette phase, chaque nœud recevra le signal beacon de tous ses voisins. Dans la deuxième phase, Chaque nœud envoie une déclaration de clusterhead avec le niveau de puissance nécessaire pour atteindre son plus proche voisin(s), puis il augmente son niveau de puissance étape par étape jusqu'à ce qu'il atteigne tous ses voisins dont il a reçu un message lors de la phase précédente. Dans cette phase, chaque nœud déterminera son niveau de puissance d'adaptation sur cette greedy heuristique. Simultanément, chaque nœud stockera le niveau de puissance nécessaire pour atteindre ses voisins. Dans la dernière phase, grâce à la beacon phase, chaque nœud possède les informations concernant la puissance résiduelle de ses voisins. Un nœud u est élu clusterhead, si u possède la plus grande puissance résiduelle parmi tous ses voisins. Bien que cet algorithme permet de prolonger la durée de vie du réseau, il nécessite plusieurs phases pour construire la structure. Cela augmente le trafic réseau et gaspille la bande de passante du réseau. Cet algorithme construit également des clusters recouvrants.

Les algorithmes que nous avons présentés jusqu'à maintenant utilisent une seule métrique pour construire les clusters, d'autres algorithmes utilisant une combinaison des métriques ont été proposés dans la littérature. Les métriques utilisées peuvent être le degré du nœud, la mo-

2. Global Position System

bilité, la puissance de transmission, l'énergie, la densité, le poids, etc. Le but de ces algorithmes est de construire des clusters adaptés à chaque type d'application. Selon le type d'application visé, tous où une partie de différents paramètres peuvent être utilisés pour le choix du clusterhead. Par exemple, dans un réseau de capteurs où l'énergie est une ressource précieuse, il est nécessaire de faire associer à la métrique l'énergie résiduelle avec un coefficient élevé.

Dans [CDT02], les auteurs ont proposé un algorithme de clustering appelé Weighted Clustering Algorithm (WCA). L'algorithme WCA utilise quatre métriques pour estimer le poids d'un nœud : la différence de degré Δ_u, la somme des distances avec ses voisins D_u, la mobilité relative M_u et le temps de service P_u au cours de laquelle un nœud u est resté en tant que clusterhead. Le poids d'un nœud u est calculé selon l'équation suivante :

$$W_u = w_1\Delta_u + w_2 D_2 + w_3 M_u + w_4 P_u \tag{3.1}$$

$$\text{avec } w_1 + w_2 + w_3 + w_4 = 1$$

La différence de degré du nœud u notée Δ_u est la différence entre le degré de u et une valeur M représentant le nombre de nœuds qu'un clusterhead peut servir. Pour calculer la valeur de M, les auteurs supposent que le nombre de nœuds qu'un clusterhead doit servir est déterminé à l'avance, sans toutefois donner des détails la façon dont cette valeur est calculée. La mobilité relative M_u est calculée de la même manière que dans MOBIC. Les distances P_u entre le nœud u et ses voisins sont calculées en utilisant un GPS. Un nœud u est élu clusterhead, si u possède la plus petite valeur de la somme pondérée de ces métriques. Quand un nœud sort dans la zone de couverture de tous les clusterheads du réseau, le processus du clustering est relancé, ceci dit cet algorithme ne résout pas les limitations de LCA et son variant LCC dans la maintenance des clusters. Dans WCA, avant de commencer le processus du clustering, il est nécessaire de connaître le poids de chaque nœud du réseau. Cependant, les méthodes utilisées pour estimer le poids des nœuds sont assez complexes et peuvent gaspiller les ressources du réseau. En conséquence, l'overhead induit par WCA est très élevé.

Dans [AJRA08], les auteurs ont proposé un algorithme appelé Distributed Score Based Clustering Algorithm (DSBCA) dans le but de minimiser le nombre de clusters et prolonger la durée de vie du réseau. DSBCA utilise également une combinaison de quatre métriques pour calculer le score d'un nœud : autonomie de batterie B_r, le degré N_n, le nombre de nœuds membres N_m et la stabilité S. Le score d'un nœud u est calculé selon l'équation suivante :

$$Score = B_r C_1 + N_n C_2 + S C_3 + N_m C_4 \tag{3.2}$$

$$\text{où } C_1, C_2, C_3 \text{ et } C_4 \text{ sont des constantes}$$

L'autonomie de batterie B_r d'un nœud u représente l'énergie résiduelle de u. Le degré N_n est le nombre de voisins de u, le nombre de nœuds membres N_m représente le nombre de nœuds qu'un clusterhead doit servir. La stabilité B_r représente le temps total où les voisins d'un nœud u sont restés dans le voisinage de u. Plus ce temps est important, plus la stabilité est meilleure. La stabilité est calculée selon l'équation suivante :

$$S = \sum_{i=0}^{n} T_{RF} - T_{RL} \tag{3.3}$$

avec n nombre de nœuds voisins

où T_{RF} et T_{RL} représentent respectivement le moment où un nœud a reçu le premier et le dernier message de la part d'un voisin.
Chaque nœud calcule son score selon l'équation 3.2 et le diffuse à ses voisins. Un nœud u est élu clusterhead, si u a le plus grand score parmi tous ses voisins. Cependant, les auteurs n'-explicitent pas le comportement de l'algorithme en cas de modification topologique. Bien que DSBCA génère moins de clusters que WCA mais il ne résout pas les limitations de WCA.

Comme nous l'avons vu, il existe dans la littérature de nombreux algorithmes de clustering à 1 saut, i.e chaque nœud du réseau est à 1 saut de son clusterhead. Parmi ces algorithmes, certains construisent des clusters recouvrants (un nœud peut appartenir à deux ou plusieurs clusters). Les structures sont ensuite utilisées pour faire du routage. Dans ce type de structure, les clusterheads et les nœuds de passage sont chargés de coordonner le routage dans le réseau. Cette technique permet de ne pas solliciter tous les nœuds pour la fonction du routage. Les techniques les plus récentes construisent des clusters non-recouvrants. Ces clusters sont plus stables comparativement à ceux construits par les autres techniques. Ce type de clusters permettent également la réutilisation spatiale de fréquences ou les codes CDMA [3] (les nœuds appartenant aux clusters non adjacents peuvent utiliser le même code CDMA). Cependant, comme nous avons pu le constater précédemment, la plus part de ces algorithmes ne sont pas auto-stabilisants et utilisent d'autres techniques pour la maintenance de clusters.

Par la suite d'autres approches permettant la construction des clusters à k sauts sont apparues. Dans ce type de clusters, chaque nœud est à k sauts de son clusterhead.

3.2.2 Approches générant des clusters à k sauts

Les algorithmes de clustering à k sauts permettent de construire des clusters où chaque nœud est au maximum à k sauts de son clusterhead. Ces algorithmes sont souvent des extensions des algorithmes de clustering à 1 saut.

Dans [GN+02], les auteurs ont généralisé l'algorithme de Lin et Gerla [LG97] afin de construire des clusters à k sauts. Ils supposent que chaque nœud a la connaissance de son voisinage à k sauts de lui. Un nœud ayant le plus petit identifiant parmi tous ses voisins à distance k de lui s'élit clusterhead . Ensuite, il diffuse son statut de clusterhead à ses voisins à k sauts. Un nœud u peut décider de s'élire clusterhead ou de s'attacher à un clusterhead que si tous ses voisins à k sauts ayant un plus petit identifiant que lui ont diffusé leur décision d'être clusterhead ou de s'attacher à un clusterhead. Ainsi, si aucun de ces nœuds est déclaré clusterhead alors u se déclare lui même clusterhead et diffuse cette décision à ses voisins à k sauts. Sinon u s'attache au clusterhead ayant le plus petit identifiant (s'il y en a plusieurs bien sûr). Afin de minimiser les clusters, les auteurs proposent d'utiliser le degré du nœud au lieu de l'identité. Ainsi, le nœud ayant la plus haute connectivité à k sauts est élu clusterhead. En cas d'égalité, l'identifiant du nœud est utilisé pour les départager. Cependant, en cas de modification topologique il nécessite une reconstruction entière de la structure. Cet algorithme construit des clusters recouvrants

3. Code Division Multiple Access

(un nœud peut appartenir à deux ou plusieurs clusters).

Dans [ASK02], les auteurs ont proposé d'utiliser une métrique particulière nommée "associativité". Le but est de privilégier les nœuds stables dans le choix du clusterhead. Ce critère de sélection du clusterhead a l'effet de choisir les nœuds les plus stables. Chaque nœud comptabilise le temps dont chacun de ses voisins reste dans son voisinage et fait la somme sur chaque voisin. Pour cela, les nœuds génèrent périodiquement des messages de contrôle afin de montrer leur existence aux nœuds voisins. Quand un nœud reçoit un tel message, il met à jour la valeur d'associativité. A chaque réception, un nœud u vérifie ses voisins actuels déjà présents lors de la période précédente et incrémente la valeur d'associativité associée à chacun d'eux. Quand un nœud disparaît, sa valeur d'associativité est remise à zéro. Quand un nœud apparaît, il prend la valeur 1. L'associativité d'un nœud u correspond à la somme des valeurs d'associativité associées à chacun de ses voisins. Une grande valeur d'associativité d'un nœud u indique un état de faible mobilité dans son voisinage. Un nœud u est élu clusterhead, s'il possède une grande valeur d'associativité dans son voisinage à k sauts et un degré supérieur à une valeur fixée à l'avance. Les clusters construits sont des clusters à k sauts et la métrique choisie apporte plus de stabilité que l'identifiant du nœud ou son degré. Cet algorithme construit également des clusters recouvrants.

Dans [FM02], les auteurs ont proposé une approche pour construire des clusters à k sauts. cet algorithme fonctionne en deux étapes. La première étape consiste à construire un arbre couvrant du réseau en utilisant l'algorithme proposé dans [AjWF02] afin de construire un ensemble dominant connecté de cardinalité minimale (MCDS) (voir 4.2.4). La seconde étape de l'algorithme consiste à partitionner l'arbre couvrant en des ensembles disjoints S_i, où S_i est un 2k-sous-arbres. Un 2k-sous-arbres est un arbre de diamètre au plus $2k$ sauts. Le diamètre d'un sous arbre S_i est noté comme suit :

$$D(S_i) = max\{d(u,v)/u,v \in S_i\} \tag{3.4}$$

Chaque S_i consiste en un cluster à k sauts. Cependant, la construction d'une telle structure génère un trafic important qui se répercute sur le temps de convergence. De plus la complexité en message de l'algorithme est $O(n)$, où n est le nombre de nœuds du réseau. En plus, les auteurs n'abordent pas la maintenance des clusters construits en cas de changement topologique.

Dans [APV+00], les auteurs ont proposé une heuristique appelé Max-Min D-cluster afin de construire des clusters à D sauts non-recouvrants, où D est le paramètre de l'heuristique. Le nombre de clusters construits est fonction du paramètre D, c'est à dire plus la valeur du paramètre D est importante, moins on aura de clusters. L'algorithme utilise l'identifiant du nœud pour le choix du clusterhead. L'algorithme se décompose en trois phases. Lors de la première phase, chaque nœud collecte les identifiants de ses voisins à D sauts et garde le plus grand appelé aussi "WINNER". Puis, il diffuse de nouveau le "WINNER" à ses voisins à D sauts lors de la deuxième phase. Au cours de la deuxième phase, chaque nœud garde le plus petit des identifiants "WINNER" qu'il reçoit lors de cette phase (le plus petit parmi les plus grands). La troisième étape consiste au choix du clusterhead qui est basé sur les identifiants des nœuds enregistrés lors de deux phases précédentes. Un nœud u sera élu clusterhead, s'il est choisi par un nœud comme "WINNER" lors de la deuxième phase. Sinon, si le nœud u a reçu "WINNER" durant chacune des deux phases 1 et 2, alors il élit le nœud dont l'identifiant correspond

à la valeur "WINNER" comme clusterhead. Sinon, u élit comme clusterhead le nœud ayant le plus grand identifiant parmi son voisinage à D sauts. Cependant, cet algorithme nécessite un échange d'informations assez important et augmente de façon considérable l'overhead du réseau.

Dans [AP00], les auteurs ont proposé une version améliorée de l'algorithme Max-Min D-cluster. Le but est d'apporter une équité entre les nœuds et donner ainsi à chaque nœud l'opportunité de servir comme clusterhead. Par exemple quand un nœud reste longtemps comme clusterhead, il épuise rapidement ses ressources (batterie par exemple). L'algorithme utilise l'identifiant virtuel (VID) comme métrique dans le choix du clusterhead. Initialement, l'identifiant virtuel d'un nœud u correspond à son propre identifiant ($VID_u = id_u$). Après chaque processus d'élection de clusterhead, chaque nœud non clusterhead incrémente de 1 son identifiant virtuel ($VID_u = VID_u + 1$) jusqu'à atteindre une valeur seuil appelée Max_{count}. Le nœud ayant le plus grand VID dans son voisinage à k sauts devient clusterhead. En cas d'égalité, le nœud qui a servi le moins en tant que clusterhead sera élu clusterhead. S'il y a toujours égalité alors le nœud qui a le plus grand identifiant sera élu clusterhead. Un nœud garde son statut de clusterhead pendant une période de temps t, après cette période, son VID passe à zéro et il abandonne son statut de clusterhead. Lorsque deux clusterheads deviennent voisins, le clusterhead ayant le plus petit VID abandonne son statut de clusterhead. Les auteurs proposent également d'utiliser le degré du nœud comme VID dans le choix du clusterhead. Bien que cet algorithme apporte une certaine stabilité de la structure construite, il nécessite une synchronisation des nœuds afin qu'ils incrémentent leurs VID et comptabilisent la période durant laquelle ils ont gardé leur statut de clusterhead. Ainsi, cette synchronisation est coûteuse en termes de nombre de messages échangés.

Dans [LC00], les auteurs ont proposé un algorithme qui est basé sur aucune métrique dans le choix du clusterhead. Il s'exécute de la façon suivante. Lorsqu'un nœud u arrive dans le réseau, il est en phase d'initialisation. Il consulte ses voisins s'ils sont en phase d'initialisation ou s'ils sont déjà rattachés à un clusterhead, dans ce cas à quelle distance se situe ce clusterhead. Si tous les voisins de u sont en phase d'initialisation, alors le nœud u va s'élire clusterhead et diffuse son statut de clusterhead à ses voisins. Tous les voisins à k sauts de u qui n'ont aucun clusterhead plus proche que u s'attacheront au cluster formé par u. Sinon le nœud u s'attache au cluster de son voisin dont le clusterhead est le plus proche et qui se trouve au plus k sauts de lui. Si tous les clusterheads de ses clusters voisins sont à plus de k sauts de u, alors u se déclare lui même clusterhead et recrute tous ses voisins à moins de k sauts qui ont des clusterheads plus éloignés que u.
Dans cet algorithme, si deux clusterheads se retrouvent à moins d'une distance D, avec D inférieur à k ($D < k$). Le clusterhead ayant le plus petit identifiant renonce à son statut de clusterhead et tous ses membres doivent trouver d'autres clusterheads. Cet algorithme construit des clusters non-recouvrants et deux clusterheads sont éloignés d'au moins $k + 1$ sauts. Cependant, l'opération de maintenance de la structure s'avère coûteuse, cela est dû au fait que quand un nœud abandonne son statut de clusterhead, ses membres se retrouveront sans clusterheads et cela entraîne une reconstruction de la structure.

Dans [KVCP97], les auteurs ont proposé un algorithme générant des clusters à k sauts. Les auteurs utilisent la distance entre les nœuds comme critère dans le choix du clusterhead.

Dans un cluster, les nœuds sont distants d'au plus k sauts. Ainsi, chaque nœud nécessite la connaissance de son voisinage à k sauts. Les clusters construits sont sans clusterhead et ils sont recouvrants. Un nœud appartenant à plusieurs clusters est dit nœud frontière. Cependant, cet algorithme exige que chaque nœud maintient trois tables : une table de voisinage, une table contenant la liste des clusters du réseau et une table contenant les nœuds frontières. Malheureusement, cela peut être coûteux en termes de mémoire et de messages échangés.

Dans [NM04], les auteurs ont proposé d'utiliser une nouvelle métrique appelée densité. Le but est de minimiser la reconstruction de la structure quand un petit changement topologique se produit dans le voisinage d'un nœud. La densité d'un nœud u est définie par :

$$\rho_k(u) = \frac{|e = (v, w) \in E|\ w \in \{u, \Gamma_k(u)\}\ et\ v \in \Gamma_k(u)}{\delta_k(u)} \tag{3.5}$$

$\Gamma_k(u)$ représente le k voisinage du nœud u et $\delta_k(u) = |\Gamma_k(u)|$. Le k voisinage d'un nœud u est l'ensemble des nœuds voisins à k sauts de lui.

Chaque nœud calcul sa densité et diffuse à ses voisins. Ainsi le nœud ayant la plus grande densité dans le voisinage est élu clusterhead. En cas d'égalité, le nœud ayant le plus petit identifiant sera élu. Dans cet algorithme, si un nœud u choisit le nœud v comme clusterhead, v peut aussi choisir le nœud w comme clusterhead et ainsi de suite. Ainsi, un cluster grandit jusqu'à atteindre les frontières d'un autre cluster. La seule contrainte introduite dans cet algorithme est que deux clusterheads ne peuvent pas être voisins. Cela permet d'éviter qu'un clusterhead soit trop excentré dans son cluster. Le calcule de la densité nécessite la connaissance des voisins à $k + 1$ sauts. Pour la maintenance de la structure, chaque nœud calcule périodiquement sa mobilité et sa densité. Si le nœud est stable, alors il compare sa densité à celle de ses voisins et choisit comme clusterhead le nœud ayant la plus grande densité. Par contre, si le nœud est trop mobile, il ne s'attachera à aucun cluster. Cet algorithme construit des clusters non-recouvrants.

Dans [YWC05], les auteurs ont proposé une extension de l'algorithme de plus petit identifiant afin de construire des clusters à k sauts. Ils supposent que chaque nœud a la connaissance de son voisinage à k sauts. Les nœuds qui ont la plus grande priorité parmi tous leurs voisins à k sauts s'élisent clusterheads et diffusent leur statut de clusterhead à leurs voisins à k sauts. Chaque nœud non clusterhead collecte ces informations et sélectionne un cluster pour être membre de ce cluster. Dans l'hypothèse ou un nœud a deux ou plusieurs voisins clusterheads à k sauts, plusieurs techniques sont utilisées pour permettre aux nœuds de choisir un unique cluster. Ainsi, nous avons entre autre : la technique basée sur l'identifiant, sur la distance et sur la taille du cluster. Avec la technique basée sur l'identifiant, les nœuds choisissent le clusterhead ayant le plus petit identifiant. Si la technique basée sur la distance est utilisée alors les nœuds choisissent le clusterhead le plus proche. Avec la technique basée sur la taille du cluster, la décision est prise en tenant compte de la taille du cluster. Cependant, en cas de modification l'algorithme nécessite la reconstruction entière de la structure. Cet algorithme construit des clusters non recouvrants.

Dans [Pel00], l'auteur a proposé un algorithme qui partitionne un graphe en clusters à k sauts. Cet algorithme est itératif, à chaque itération construit un cluster. Il fonctionne selon le principe suivant : il sélectionne arbitrairement un sommet u de V et ajoute à chaque itération

une couche autour de u. Les sommets ajoutés au cluster u sont retirés de V. Ce processus se répète jusqu'à ce que le paramètre k soit atteint. Cet algorithme construit des clusters disjoints.

Dans [DMZ10, DMZ06, DM04], les auteurs ont proposé des algorithmes déterministes qui partitionnent un graphe en clusters. Ces algorithme s'exécutent de façon distribuées et choisissent un nœud u du graphe comme leader qui sera le centre du nouveau cluster. Ces algorithmes sont également itératifs et à chaque itération ajoutent une couche autour du cluster u. Contrairement à l'algorithme précédent, ces algorithmes construisent des clusters en parallèle à différents endroits du graphe. Ils construisent des clusters disjoints.

3.3 Notre solution

Comme nous l'avons vu précédemment, il existe dans la littérature plusieurs solutions pour structurer un réseau. Nous avons les algorithmes de clustering à 1 saut qui sont beaucoup plus développés et les algorithmes de clustering à k sauts. Ainsi, les algorithmes à k sauts s'inspirent généralement des algorithmes à 1 saut. Cependant, la plupart de ces algorithmes ne répondent pas a la totalité des exigences que nous avons introduit dans la section 3.1.3. Partant de ce constant, nous avons proposé un nouvel algorithme déterministe de clustering à 1 saut qui s'adapte aux modifications topologiques du réseau.

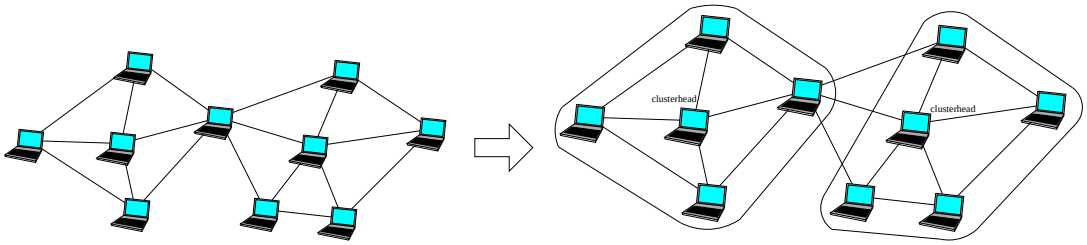
Notre principal objectif est de hiérarchiser le réseau pour le structurer, cette approche permet de rendre l'utilisation du réseau plus efficace, par exemple le routage, la diffusion, la localisation, etc.

Ainsi, nous proposons une structure basée en clusters pour hiérarchiser le réseau. Cette structure doit présenter une utilité par exemple pour le routage, comme nous le verrons par la suite. L'algorithme que nous proposons est un algorithme auto-stabilisant qui est basé sur des connaissances locales pour construire les clusters. Il s'adapte lui même à la topologie du réseau, qui évolue de façon aléatoire au cours du temps. La construction des clusters se fait à l'aide des identités portées par chaque nœud, identité supposée unique. Comme l'identifiant des nœuds étant inchangeable, il apporte plus de stabilité par rapport à certaines métriques comme par exemple le degré des nœuds. Avec notre solution, l'ensemble des phases de la découverte de topologie et la structuration du réseau ne nécessite qu'un unique message. Cette approche consiste donc à combiner les deux phases en une seule et à fabriquer des clusters disjoints, c'est à dire qu'un nœud n'appartient qu'à un seul cluster. Elle permet donc d'optimiser les messages échangés.

Nous allons dans un premier temps exposer les motivations de notre solution. En suite, nous présenterons dans la section 3.5 la méthodologie et notations adoptées, puis dans la section 3.6 nous détaillerons l'algorithme de construction des clusters. Dans la section 3.7 nous présenterons les preuves de notre solution. Enfin nous présenterons dans la section 3.8 les résultats de simulations.

3.4 Motivations

Dans cette partie, nous rappelons rapidement les buts recherchés de notre solution
- Structurer le réseau pour améliorer les communications,
- Minimiser le nombre de messages dans le réseau,
  - Messages de contrôle,
  - Routage.
- Émergence d'une structure globale a partir d'informations locales,
- Permettre le passage à l'échelle,
- Supprimer les tempêtes de broadcast dans le réseau.



(a) Réseau non hiérarchisé (b) Réseau hiérarchisé

FIGURE 3.4 – Schéma de la construction des clusters

La figure 3.4 présente un aperçu général de notre structure hiérarchique. Dans un premier temps, les nœuds initient la découverte de voisinage en envoyant un message de type *hello*. Ensuite, les nœuds construisent la structure en utilisant les informations collectées. Bien que la description ici est itérative, les différentes phases de l'algorithme s'exécutent en parallèle afin d'optimiser les messages échangés et réduire le temps de convergence. Nous détaillerons le fonctionnement de l'algorithme dans les sections suivantes.

3.5 Méthodologies et notations

Un réseau ad hoc peut être modélisé par un graphe non orienté $G = (V, E)$ où V est l'ensemble des nœuds du réseau et E modélise l'ensemble des connexions qui existent entre ces nœuds. Une arête $(u, v) \in E$ si et seulement si u et v peuvent mutuellement recevoir les transmissions de u et v. Ce qui implique que tous les liens entre les nœuds sont bidirectionnels. Dans ce cas, on dit que u et v sont voisins. L'ensemble des voisins d'un nœud $u \in V$ est noté N_u. Chaque nœud u du réseau à un identifiant unique id_u et peut communiquer avec un sous-ensemble $N_u \subseteq V$.

Définition 3.1 (Cluster) *On définit un cluster par un sous-graphe connexe du réseau, dont le diamètre est inférieur ou égal à 2. L'ensemble des nœuds du cluster i est noté V_i.*

Définition 3.2 (Identifiant du cluster) *Le cluster possède un identifiant correspondant à l'identité la plus grande des nœuds du cluster. L'identifiant du cluster i est noté cl_i.*

Chaque nœud possède l'identifiant du cluster noté cl et tous les nœuds d'un même cluster possède le même identifiant de cluster.

Définition 3.3 (Statut d'un nœud) *Chaque nœud u possède un statut, noté* $statut_u$*, qui peut prendre l'une des valeurs suivantes :*
- CH *: clusterhead,*
- NM *: nœud membre,*
- NP *: nœud de passage.*

Ces 3 rôles sont définis de la façon suivante :

Définition 3.4 (Clusterhead) *Un nœud u est dit clusterhead ssi* $id_u = max(id_v / v \in N_u \wedge cl_u = cl_v)$

L'identifiant d'un cluster correspond à l'identité du clusterhead et, pour tout nœud u du cluster, cl_u est égal à l'identifiant du clusterhead.

Définition 3.5 (Nœud membre) *Un nœud u est dit nœud membre ssi* $(\forall\ v \in N_u,\ cl_v = cl_u \wedge id_u \neq cl_u)$

Définition 3.6 (Nœud de passage) *Un nœud u est dit nœud de passage ssi* $(\exists\ v \in N_u,\ cl_u \neq cl_v)$*.*

Un nœud de passage contrairement à un nœud membre, possède un rôle particulier. Il permet d'accéder à un ou plusieurs clusters voisins.

On remarque que le statut et l'identifiant de cluster du nœud est déterminé en fonction de sa vision de l'état de ses voisins (l'identifiant du nœud et du cluster, et le statut). L'état de ses voisins est échangé au cours de l'exécution de l'algorithme et stocké dans un ensemble noté N_u.

Un nœud u possède les données suivantes :
- N_u : désigne l'ensemble des voisins d'un nœud u et leur état (id, $statut$, cl_u),
- cl_u : désigne le cluster id du nœud u,
- id_u : désigne l'identité du nœud u,
- $statut_u$: statut du nœud u ; $statut_u \in \{CH, NM, NP\}$.

3.6 L'algorithme

Dans cette section nous présentons un algorithme de clustering (voir l'algorithme 1) qui est un algorithme auto-stabilisant. Nous allons tout d'abord commencer par présenter le fonctionnement intuitif de l'algorithme, avant de le présenter formellement.

3.6.1 Son principe d'exécution

Comme décrit dans la définition 1, le choix du clusterhead est basé sur les identités associées à chaque nœud. Le clusterhead est le nœud qui a la plus grande identité parmi tous ses voisins, dans son cluster. Nous aurions également pu choisir le nœud de plus petite identité pour notre clusterhead. Ce choix ne change pas les caractéristiques de notre solution. Afin de mieux répondre aux exigences imposées par le réseau sans fil notamment la mobilité des nœuds, l'algorithme satisfait les 2 propriétés suivantes :
- Chaque nœud du réseau doit appartenir à un unique cluster,

– Tous les nœuds d'un même cluster sont à distance au plus de 1 de leur clusterhead.

Nous supposons également que la propriété suivante est vérifiée par le réseau :

– Un message envoyé par un nœud est reçu correctement en un temps fini par sa destination.

L'algorithme partitionne le réseau en clusters et chaque cluster a un unique clusterhead. La construction des clusters se fait par échange périodique de messages que nous appellerons, des messages *hello*. Chaque nœud du réseau échange avec ses voisins, ces messages *hello*. Chaque message *hello* transmis par un nœud u contient trois valeurs qui sont : id_u, $statut_u$ et cl_u. Ce message sert également à chaque nœud pour vérifier la présence de ses voisins. Ainsi, si un nœud ne reçoit plus de message *hello* de la part d'un de ses voisins, à la fin d'une période, il considère que ce voisin a disparu. Donc chaque nœud attend une période déterminée à l'avance et on suppose que durant cette période, tous les nœuds ont envoyé leur message *hello*.

A la fin de cette période, chaque nœud compare les identités des expéditeurs. Le nœud dont l'identité est la plus grande dans un voisinage sera clusterhead. Il utilisera cette information dans son prochain message *hello*, afin d'informer son voisinage de son statut. A la première réception d'un message *hello* qui contient un $statut = CH$, les nœuds voisins deviendront des nœuds membres, c'est à dire avec $statut = NM$. Dans le cas où un nœud a dans son voisinage plusieurs clusterheads, il deviendra un nœud de passage, c'est à dire avec $statut = NP$. A la fin du processus chaque nœud sera dans un des trois états suivants : clusterhead, nœud membre ou nœud de passage.

Dans l'hypothèse où deux clusterheads deviennent voisins, celui dont l'identité est la plus petite deviendra nœud membre de l'autre nœud.

Exemple 3.1 *La figure suivante montre la construction de clusters suivant notre algorithme. Nous obtenons* 3 *clusters. Le cluster B possède l'identité* $cl_B = 13$ *qui correspond à la plus grande des identités de* $V_B = \{1, 5, 6, 10, 13\}$. *Il possède deux nœuds de passage qui permettent d'atteindre les clusters A et C. Les nœuds* 1 *et* 10 *sont des nœuds membres car tous leurs voisins appartiennent à leur cluster.*

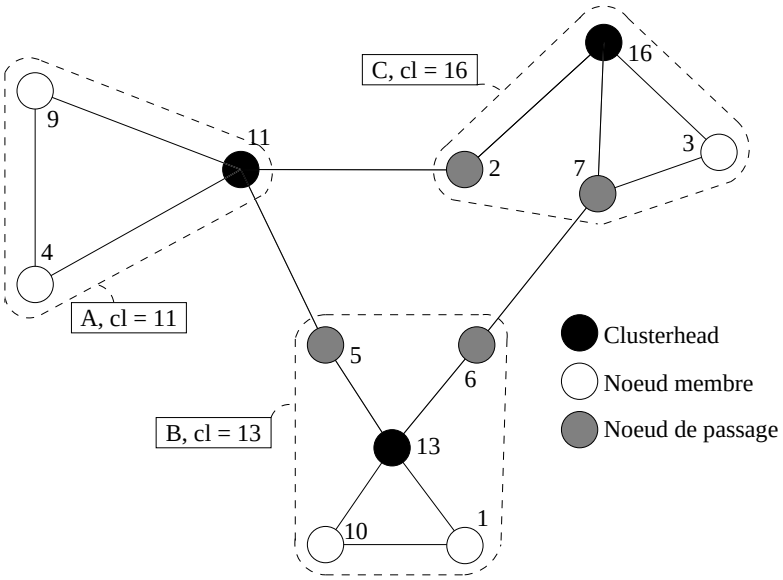
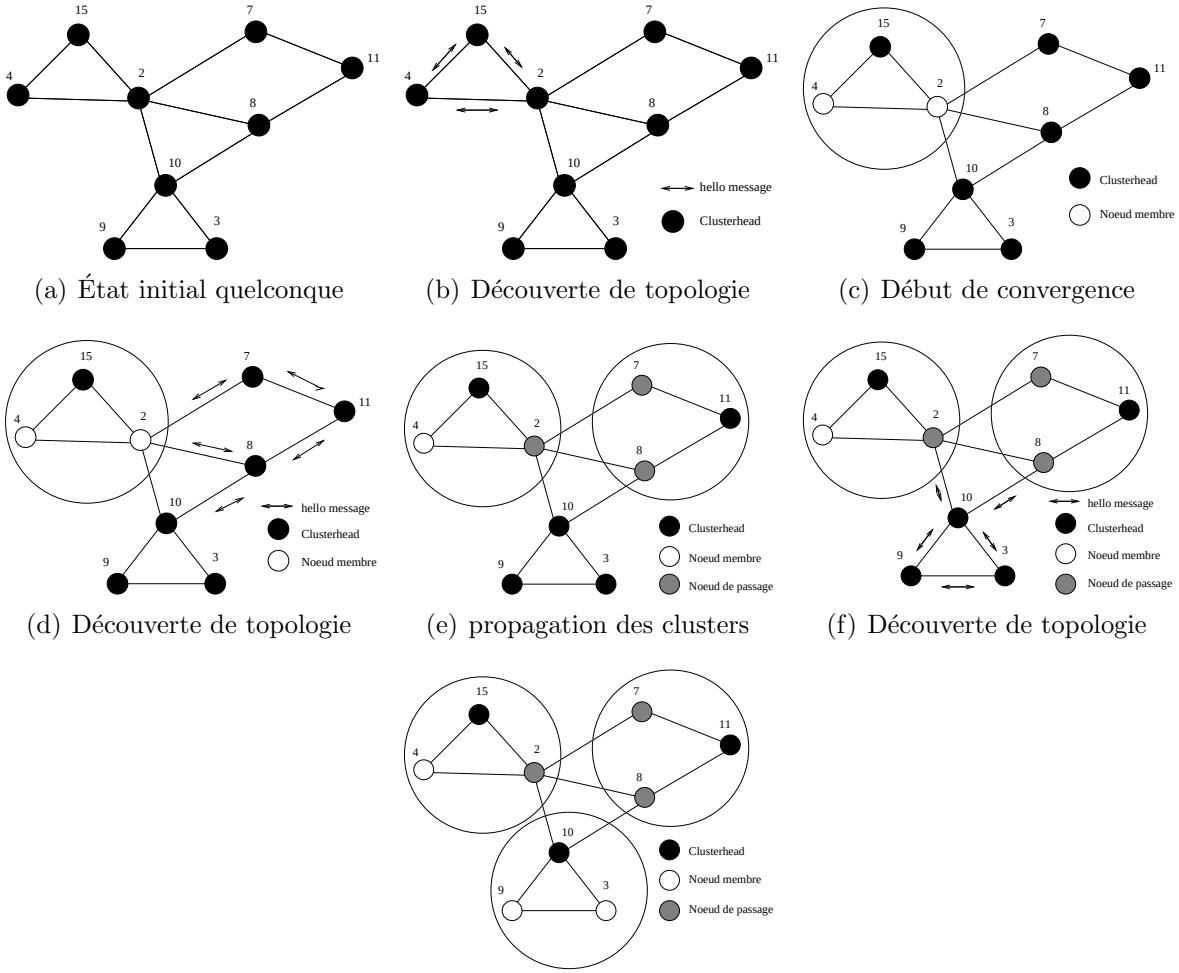


FIGURE 3.5 – Déclaration des clusterheads

3.6.2 Exemple détaillé d'exécution

Nous détaillons ici un exemple d'exécution de notre algorithme sur un graphe de 9 nœuds. Étant donné que l'algorithme est auto-stabilisant, il garantit qu'à partir d'une configuration arbitraire, l'algorithme converge vers un état dans lequel le réseau sera entièrement organisé en clusters.

Soit une configuration où tous les nœuds du réseau sont clusterheads. Dans la figure 3.6(b), les nœuds 15, 4 et 2 envoient un message *hello* pour découvrir leurs voisins. Après avoir collectées les informations, chacun est alors en mesure de comparer son identité à celle de ses voisins. A partir de là, le nœud dont l'identité est la plus grande conserve son statut de clusterhead et ses voisins deviennent à nouveau nœuds membres. Dans notre cas, le nœud 15 conserve son statut et les nœuds 4 et 2 redeviennent nœuds membre. De la même façon, les autres nœuds du réseau envoient un message *hello*. Après cette découverte, comme montré dans la figure 3.6(g), les nœuds 10 et 11 conservent leur statut de clusterhead. Les nœuds reliés aux clusters adjacents deviennent nœuds de passage, c'est le cas des nœuds 2, 7 et 8. Le réseau est maintenant dans une configuration normale.



(a) État initial quelconque (b) Découverte de topologie (c) Début de convergence

(d) Découverte de topologie (e) propagation des clusters (f) Découverte de topologie

(g) Convergence de l'algorithme

FIGURE 3.6 – Illustration du processus de convergence de l'algorithme

3.6.3 Maintenance des clusters

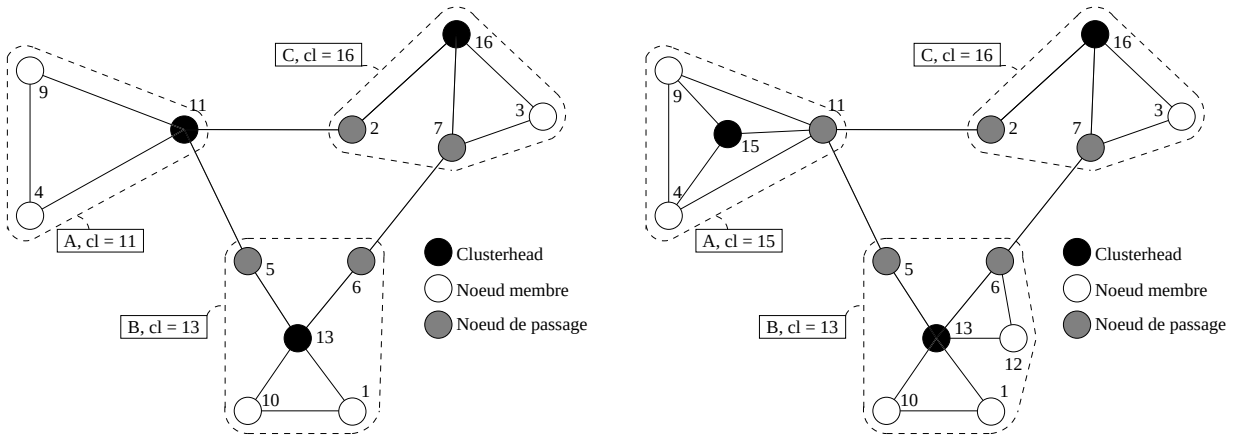
Dans les réseaux ad hoc, la topologie du réseau change fréquemment due à la mobilité des nœuds. Nous devons donc gérer :

- Les nœuds qui apparaissent,
- Les nœuds qui disparaissent,
- Les nœuds qui se déplacent.

Notre algorithme permet également de gérer ces cas.

- *L'apparition d'un nouveau nœud.* Quand un nouveau nœud arrive dans le réseau, il diffuse à un intervalle de temps régulier un message de type *hello* et collecte les identités de ses voisins. S'il n'y a aucun clusterhead dans le voisinage du nouveau nœud, celui-ci devient clusterhead. Éventuellement, des voisins se rattacheront à son cluster si l'identité de leur clusterhead est inférieure à celle du nouveau nœud. S'il existe un clusterhead dans son voisinage, soit 1) l'identité de ce clusterhead est plus grande que son identité ou 2) l'identité du clusterhead est inférieure à son identité. Dans le cas 1), le nouveau nœud

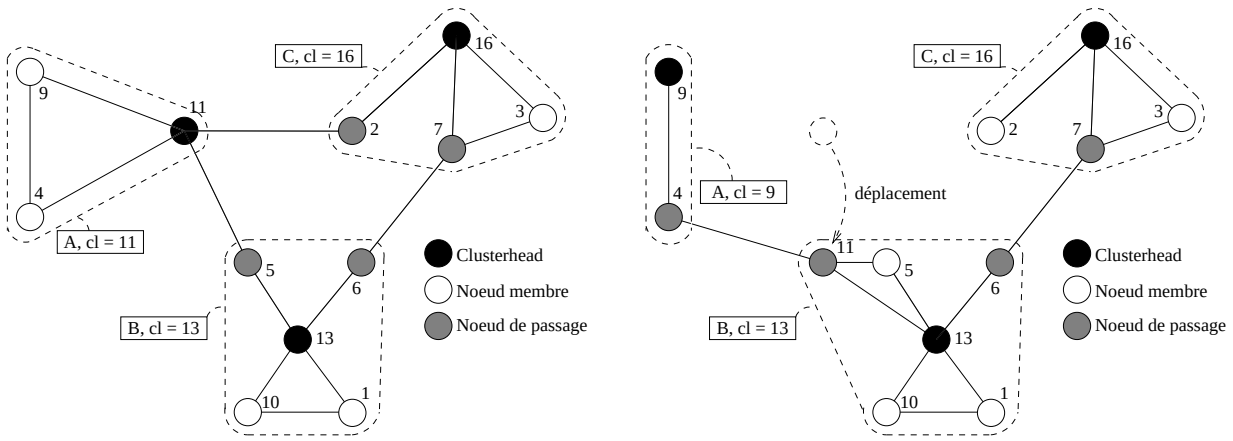
se rattache au clusterhead. Si tous ses voisins appartiennent à ce cluster, il est nœud membre. Par contre, il devient nœud de passage. Dans le cas 2), il devient clusterhead et l'ancien clusterhead se rattachera à lui.
Dans la figure 3.7(b), le nœud 15 apparaît. Il est dans le cas 2) : il possède une meilleure identité que le clusterhead 11. Le nœud 11 devient nœud de passage car il possède un voisin appartenant à un autre cluster. De même, le nœud 12 apparaît. Il possède une identité inférieure au clusterhead 13, c'est donc le cas 1),



(a) Avant l'apparition (b) Après l'apparition

FIGURE 3.7 – Convergence après apparition des nœuds

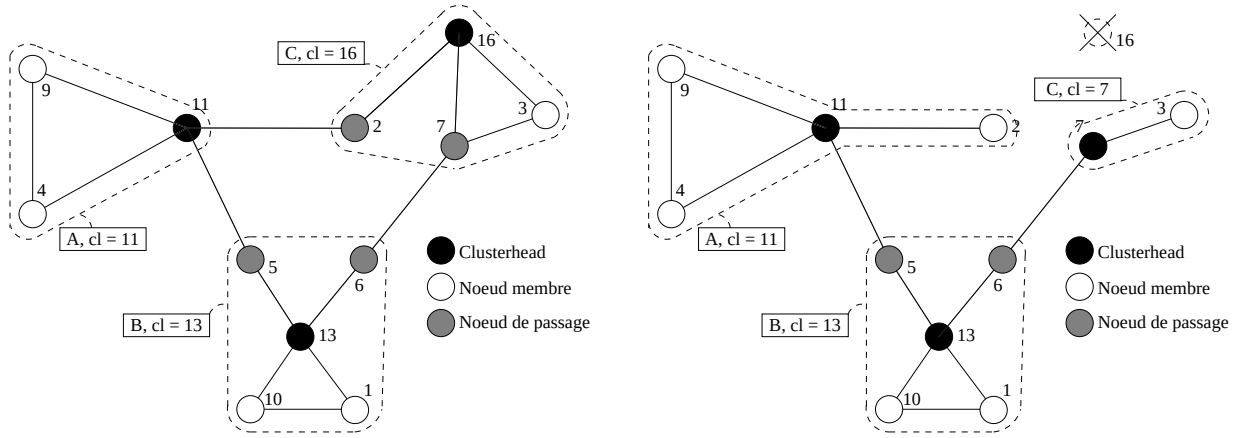
– *Le déplacement d'un nœud*. De la même manière que pour l'apparition, par échange de messages *hello*, les voisins de ce nœud s'apercevront de ce déplacement. Dans la figure 3.8(b), suite à un déplacement, le nœud 11 a perdu son statut de clusterhead. Par échange de messages *hello*, le nœud 9 s'aperçoit que le nœud 11 ne peut plus être son clusterhead. Donc le nœud 9 redevient lui même clusterhead,



(a) Avant le déplacement (b) Après le déplacement

FIGURE 3.8 – Convergence après un déplacement d'un nœud

– *La disparition d'un nœud.* Toujours par échange de messages *hello*, les voisins de ce nœud s'apercevront de cette disparition. Dans la figure 3.9(b), suite à la disparition du nœud 16, le nœud 7 redevient clusterhead. Le nœud 2 s'attache au cluster 11 et redevient nœud membre du cluster 11.



(a) Avant la disparition (b) Après la disparition

FIGURE 3.9 – Convergence après une disparition d'un nœud

Nous avons montré ici à l'aide d'un exemple comment notre algorithme s'adapte aux changements topologiques.

3.6.4 Présentation de l'algorithme

Chaque nœud exécute périodiquement l'algorithme 1 qui se décompose en trois étapes. Dans un premier temps, chaque nœud vérifie la cohérence de ses données locales (algorithme 2). Ensuite, chaque nœud diffuse un message *hello* à ses voisins et collecte les identités et les états de ses voisins (algorithme 3) via les messages *hello* reçus. Enfin, le nœud exécute l'algorithme 4 pour mettre à jour son état et ceux de ses voisins.

Définition 3.7 (Cohérence d'un nœud) *Un nœud u est cohérent ssi les propriétés suivantes sont vérifiées :*
– $statut_u = CH \Leftrightarrow id_u = cl_u$,
– $statut_u \in \{NP, NM\} \Leftrightarrow \exists\ v \in N_u\ /\ statut_v = CH \wedge cl_u = cl_v$.

Remarque : La cohérence d'un nœud dépend de la vision qu'il a de ses voisins et cette vision est échangée via les messages *hello*. Il est possible qu'il devienne incohérent suite à la réception ultérieure de messages.

L'algorithme 2 permet de corriger les incohérences sur les nœuds. Si un nœud est clusterhead et que son cluster id n'est pas sa propre identité, il corrige son cluster id en plaçant sa propre identité (règle R1.a). De même, si un nœud n'est pas clusterhead mais que son cluster id est son identité, ou qu'aucun nœud voisin ne possède l'identité du cluster id, ou encore que le clusterhead supposé n'a pas le statut de clusterhead, alors le nœud devient clusterhead et son cluster id devient sa propre identité (R1.b).

Algorithme 1 Algorithme général sur le nœud u

Vérification de la cohérence du nœud u (Algorithme 2)
$N_u \rightarrow \emptyset$
Diffusion locale d'un message *hello* (id_u, cl_u, $statut_u$)
Attente des messages des voisins et mise à jour (Algorithme 3)
Mise à jour de cl_u et $statut_u$ (Algorithme 4)

Algorithme 2 Vérification de la cohérence sur le nœud u

Si ($statut_u$ = CH) $\wedge$ ($cl_u \neq id_u$) **Alors**
/* *L'identité de cluster de u doit être l'identité de u (R1.a)* */
$cl_u \leftarrow id_u$
Fin Si
Si ($statut_u \neq$ CH) $\wedge$ (($cl_u = id_u$) $\vee$ ($\forall\ v \in N_u$, $id_v \neq cl_u$) $\vee$ ($\exists\ v \in N_u$, $cl_u = id_v \wedge statut_v \neq$ CH)) **Alors**
/* *Le statut est incohérent avec le voisinage, u devient clusterhead (R1.b)* */
$statut_u \leftarrow$ CH
$cl_u = id_u$
Fin Si

A chaque réception d'un message *hello*, l'identité du nœud qui a envoyé le message est ajouté à l'ensemble des voisins. Comme cet ensemble est vidé au préalable, le nœud ne garde en mémoire que les informations les plus récentes.

Algorithme 3 Réception d'un message $hello(id, cl, statut)$ du nœud v sur le nœud u (R0)

$N_u \leftarrow N_u \cup \{$v(id, cl, statut)$\}$

Si le nœud est clusterhead. Il vérifie qu'aucun de ses voisins clusterhead ne possède une meilleure identité que la sienne. Si ce n'est pas le cas, il devient nœud membre du plus grand clusterhead (règle R3).

Si le nœud n'est pas clusterhead. Il vérifie s'il possède la meilleure identité de son voisinage. Si c'est le cas, il devient CH avec comme cluster id sa propre identité (règle R2.a). Si un clusterhead possède une identité plus grande que son cluster id, il se rattache à ce nœud (regèle R2.b). Dans tous les autres cas, le nœud ne change pas de cluster. Donc, si un de ses voisins appartient à un autre cluster, il devient nœud de passage (règle R2.c) et sinon, il devient un nœud membre (règle R2.d).

3.6.5 Propriétés de l'algorithme

Afin d'expliquer formellement notre algorithme, et d'aborder les preuves, nous présentons les propriétés suivantes.

Propriété 3.1 (Cluster disjoint) *Quels que soient les clusters i et j avec* $id_i \neq id_j$, $V_i \bigcap V_j = \emptyset$

Propriété 3.2 (Clusterhead unique dans chaque cluster) *Soit i un cluster.* $\forall\ u \in V_i$, $\exists!\ v \in V_i$ *tel que* $statut_v = CH$.

Algorithme 4 Fin de période sur le nœud u

Si ($statut_u \neq$ CH) **Alors**
  Si ($\forall\ v \in N_u$, $id_v < id_u$) **Alors**
    / u possède l'identité la plus grande (R2.a) */*
    $statut_u \leftarrow$ CH
    $cl_u \leftarrow id_u$
  Sinon
    Si ($\exists\ v \in N_u$, $statut_v =$ CH $\wedge\ id_v > cl_u$) **Alors**
      / u se rattache au cluster de plus grande identité (R2.b) */*
      $statut_u \leftarrow$ NP
      $cl_u \leftarrow id_v$
    Sinon
      Si ($\exists\ v \in N_u$, $cl_v \neq cl_u$) **Alors**
        / Il y a un autre cluster dans le voisinage de u (R2.c) */*
        $statut_u \leftarrow$ NP
      Sinon
        / Tous les voisins sont dans mon cluster (R2.d) */*
        $statut_u \leftarrow$ NM
      Sinon
        / Rien à faire (R4) */*
      Fin Si
    Fin Si
  Fin Si
Sinon */* Le nœud u est CH */*
  Si ($\exists\ v \in N_u$, $statut_v =$ CH $\wedge\ id_v > id_u$) **Alors**
    / Il existe un clusterhead avec une identité supérieure à la mienne (R3) */*
    $statut_u \leftarrow$ NM
    $cl_u \leftarrow id_v$ tel que $v = Max(w \in N_u / \{statut_w = CH \wedge id_w > id_u\})$
  Sinon
    / Rien à faire (R4) */*
  Fin Si
Fin Si

Propriété 3.3 (Le clusterhead possède l'identité la plus grande) *Soit i un cluster* $\forall\ u \in V_i$, $statut_u = CH \Rightarrow \forall\ v \in V_i \backslash \{u\}$, $id_u > id_v$.

La technique de clustering que nous proposons partitionne le graphe en clusters dit "bien formés". C'est à dire que le cluster vérifie les deux propriétés suivantes :

Propriété 3.4 *Les clusters forment une partition du graphe.*

Preuve.
- Soit le nœud possède un voisin clusterhead dont l'identité est supérieure à la sienne. Dans ce cas, il appartient à ce cluster (règles R2.b, R2.c, R2.d et R3),
- Soit le nœud ne possède aucun voisin clusterhead, ou bien que les voisins clusterhead ont une identité inférieure à la sienne. Dans ce cas, il devient lui-même clusterhead (R2.a et R4).

□

Propriété 3.5 *Chaque nœud est à distance au plus de* 1 *d'un clusterhead.*

3.7 Preuve

Définition 3.8 (Configuration) *Nous disons qu'au terme d'une configuration, un nœud a reçu au moins un message de tous ses voisins et a exécuté l'algorithme 4.*

Définition 3.9 (Transition) *Nous appelons transition le passage d'une configuration* C_i *à une configuration* C_{i+1}. *Nous appelons transition i la transition qui permet de passer de la configuration* C_i *à* C_{i+1}.

Propriété 3.6 (Convergence) *Quel que soit l'état initial et sans occurrence de fautes, l'algorithme converge vers une configuration légale après un nombre fini de transitions.*

Propriété 3.7 (Fermeture) *A partir d'une configuration légale et sans occurrence de faute, le système restera toujours dans une configuration légale.*

Pour prouver ces deux propriétés, il est nécessaire de s'appuyer sur la définition du nœud fixé.

Définition 3.10 (Fixé) *Un nœud u est dit fixé si son* cl_u *ne change plus au cours de l'exécution. Nous notons fixé(u) égal à vrai si u est fixé.*

Les premiers noeuds qui se fixent sont les noeuds qui possèdent une identité plus grande que celle de leurs voisins.

Lemme 3.1 *Soit* $u \in V, \forall\ v \in N_u$ *tel que* $id_u > id_v$ *alors à* C_0, $statut_u = CH$ *et* $cl_u = id_u$.

Preuve.

1. Soit $statut_u = CH$,

- Si $cl_u \neq id_u$ donc la règle $R1.a$ peut s'exécuter et à C_0, $cl_u = id_u$,
- Si $cl_u = id_u$ alors seule la règle $R4$ peut être appliquée.

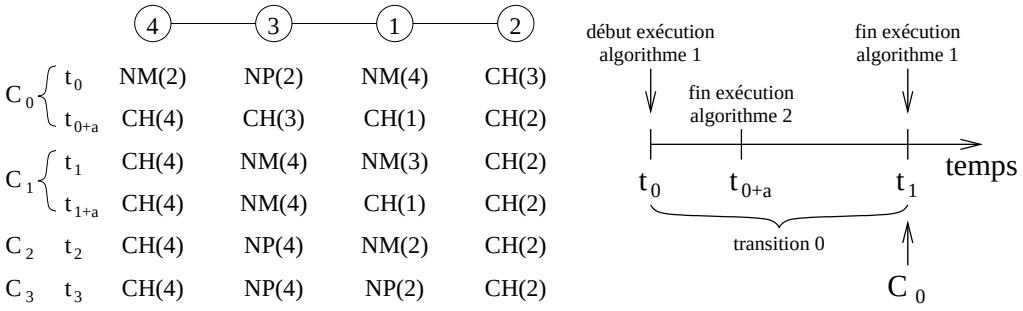
2. Soit $statut_u \neq CH$,
   - Si $cl_u = id_u$ alors la règle $R1.b$ peut s'exécuter et à C_0, $statut_u = CH$, $cl_u = id_u$,
   - Si $cl_u \neq id_u$ par définition de u, nous avons $\forall\, v \in N_u\ id_u > id_v$ donc seule la règle $R2.a$ s'exécute et à C_0, $statut_u = CH$ et $cl_u = id_u$.

□

Corollaire 3.1 *$\forall i \geq 0$ à C_i le nœud u tel que défini dans le lemme 3.1 n'exécutera que la règle $R4$ et $statut_u = CH$ et $cl_u = id_u$.*

Corollaire 3.2 *Le nœud u tel que défini dans le lemme 3.1 est fixé à C_0.*

Considérons le réseau représenté sur la figure 3.10(a), nous remarquons que les nœuds 4 et 2 possèdent les plus grandes identités dans leur voisinage et ils sont fixés à la configuration C_0.



(a) déroulement (b) Transition et Configuration

FIGURE 3.10 – Exemple d'exécution

Lemme 3.2 *Soit $v \in N_u$, u tel que défini dans le lemme 3.1 et $\forall w \in N_v$ avec $id_u > id_w$. A C_1 nous avons, $statut_v \in \{NM, NP\}$ et $cl_u = cl_v$.*

Preuve. A la configuration C_0 : $v \in N_u$

1. Soit $statut_v = CH$. Si $statut_v = CH$ alors $cl_v = id_v$. Comme $id_v < id_u$ donc seule la règle $R3$ peut s'exécuter. Donc à C_1 nous avons $Statut_v \in \{NM, NP\}$ et $cl_v \in \{\ id_u$ ou id_w avec w $\in N_v \backslash \{u\}\}$,
2. Soit $statut_v \neq CH$. Comme $id_v < id_u$ et v va recevoir un message m de la part de u avec $statut_u = CH$:
   - Si $cl_v < id_u$, d'après $R2.b$, à C_1, $statut_v = NP$ et $Cl_v = id_u$,
   - Si $cl_v = id_u$, d'après $R2.d$, à C_1, $statut_v = NM$ et $cl_v = id_u$,
   - Si $cl_v > id_u$, d'après $R2.c$, à C_1, $statut_v = NP$ et cl_v ne change pas.

□

Lemme 3.3 *Un nœud u se fixe (i.e à C_i, u n'est pas fixé, à C_{i+1} il l'est) suivant les propriétés suivantes :*

Propriété 3.8 $\forall v \in N_u, id_u > id_v$.

C'est à dire le nœud u possède la plus grande identité parmi tous ses voisins.
Preuve. Lemme 3.1 □

Propriété 3.9 $\exists v \in N_u$, *(fixé(v))* $\wedge$ $(id_v > id_u)$ $\wedge$ $(statut_v = CH)$ $\wedge$ $(\forall\ w \in N_u \setminus \{v\}$, $id_w > id_v) \wedge$ *(fixé(w))*$\wedge$ $(statut_w \neq CH$ *).*

C'est à dire il existe un nœud v voisin du nœud u dont l'identité de v est supérieure à celle de u, avec $statut_v = CH$ et tous les autres voisins w du nœud u dont les identités sont supérieures à celle de v sont fixés et ne sont pas clusterheads.

Preuve. Comme tous les voisins w du nœud u sont fixés et ne sont pas clusterheads, seule la règle $R2.d$ peut s'appliquer et à C_i, $cl_u = id_v$ □

Propriété 3.10 $\forall v \in N_u$, $(id_v > id_u) \wedge (statut_v \neq CH) \wedge$ *(fixé(v)).*

C'est à dire tous les voisins du nœud u dont les identités sont supérieures à celle de u sont fixés et ne sont pas clusterheads.

Preuve. Comme tous les voisins v du nœud u sont fixés et ne sont pas clusterheads alors, seule la règle $R2.a$ peut s'appliquer et à C_i, $cl_u = id_u$
□

A chaque configuration au moins un nœud va se fixer

Définition 3.11 (NF_i) *Nous appelons NF_i l'ensemble des nœuds fixés à C_i.*

Définition 3.12 (N_{NF_i}) *Nous appelons N_{NF_i} l'ensemble des voisins des nœuds fixés, avec $NF_i \cap N_{NF_i} = \emptyset$.*

Ce que nous voulons démontrer, c'est que $|NF_i| < |NF_{i+1}|$ et $|NF_i| = |NF_{i+1}|$ si $|NF_i| = |V|$

Définition 3.13 (Chaîne max) *Nous construisons une chaîne CM depuis le nœud u que nous notons CM(u)= (v_0, v_1, ..., v_{n-1}) telle que :*

1. $v_0 = u$ *et* $fixe(v_{n-1})$,
2. $\forall i \in [0, taille(CM)\text{-}2]$, $id_{v_i} < id_{v_i+1}$ *et* $v_{i+1} \in N_{v_i}$,
3. $\forall w \in N_{v_{n-1}}$, $(id_w < id_{v_{n-1}}) \vee$ *(fixe(w)),*
4. $\forall i \in [0, taille(CM)\text{-}2]$, $!fixe(v_i)$, *le nœud v_{n-1} est noté Max(CM(u)), la taille correspond au nombre de nœuds de la chaîne.*

Considérons le réseau représenté sur la figure 3.11, à une configuration donnée, l'ensemble des chaînes max du nœud 0 qu'on peut construire à cette configuration est : $\{(0, 1, 2), (0, 3), (0, 4, 5)\}$. $(0, 4)$ et $(0, 1)$ ne sont pas des chaînes max. Par contre, (2), (3), (5) et (4, 5) sont des chaînes max valides.

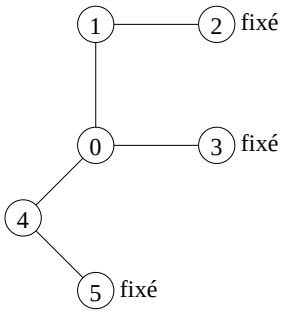


FIGURE 3.11 – Exemple des chaînes

Théorème 3.1 *Si un nœud u n'est pas fixé, il est possible de construire une chaîne max partant de u de longueur supérieure ou égale à 2.*

Preuve. Comme u n'est pas fixé, il existe un nœud v voisin de u avec $id_v > id_u$. D'après la définition de la chaîne max (définition 3.13), nous avons CM(u)= (v_0, v_1, ..., v_{n-1}) et on sait que $u = v_0$ donc on aura :
- Soit $v = v_{n-1}$ alors la taille de la chaîne max $CM(u) = 2$ et $Max(CM(u)) = v$,
- Soit $id_v < id_{v_{n-1}}$ alors la taille de la chaîne max $CM(u) > 2$ et $Max(CM(u)) \neq v$.

Dans tous les cas, la taille de la chaîne u est supérieure ou égale à 2 □

Théorème 3.2 *$\forall i \geq 0$, à C_i au moins un nœud se fixe.*

Preuve. Faisons la preuve par récurrence
- A C_0, au moins un nœud se fixe. Au pire seul le nœud possédant l'identité la plus grande du réseau se fixe d'après le lemme 3.1,
- A C_1, au moins les nœuds voisins du nœud possédant l'identité la plus grande du réseau se fixent d'après le lemme 3.2,
- A C_i, soit $u = Max(N_{NF})$, à C_{i+1}, $fixe(u)$
  Preuve par l'absurde : si à C_{i+1}, $!fixe(u)$ alors $\exists v \in N_u / id_v > id_u$ et $!fixe(v)$. Comme $!fixe(v)$, il existe une chaîne max partant de v d'après le théorème 3.1. Donc $\exists w_0, w, w_1, ..., w_{n-1}$ / $id_v < id_{w_0} < id_{w_1} < ... < id_{w_{n-1}}$.
  Or $id_v > id_u$. Donc $id_{w_{n-2}} > id_u$. Or $fixe(w_{n-2})$.
  Donc $w_{n-2} \in N_{NF}$, cela est impossible car $u = Max(N_{NF})$.

  Donc $\forall \geq 0$, à C_i, au moins un nœud se fixe

□

Corollaire 3.3 *Au maximum n transitions, tous les nœuds sont fixés.*

Théorème 3.3 *A C_i, si u est fixé et $\forall v \in N_u$, $fixe(v)$, à C_j avec $j > i$, le statut de u ne change pas.*

Preuve. Le statut dépend uniquement :
- des identités de voisins,

– des identités de clusters voisins.

Or tous les voisins de u sont fixés à C_i. Donc le statut de u ne change plus à partir de C_j avec $j > i$ □

Théorème 3.4 *A C_{n+2}, tous les nœuds sont stables.*

Preuve. A C_n, d'après le corollaire 3.3, tous les nœuds sont fixés et d'après le théorème 3.3, à C_{n+1}, tous les statuts sont fixés. Donc à C_{n+2}, tous les statuts sont échangés. □

Fermeture 3.1 *Après une transition, à partir d'une configuration légale C_i, nous obtenons une configuration C_{i+1} légale.*

Preuve. D'après le corollaire 3.3, nous savons que tous les nœuds u sont fixés et leur statut ne change plus, et en plus d'après le théorème 3.4, tous les nœuds sont stables.

Donc après la stabilisation, $\forall u \in V$, les valeurs de $statut_u$ et cl_u ne changeront plus entre C_j et C_{j+1} $\forall j$ sans faute transitoire. □

3.8 Simulations et résultats

Nous présentons dans cette partie les résultats de simulations effectués sous DASOR [4][Rab]. Les résultats présentés visent à observer la pertinence des solutions proposées. La simulation d'un système permet d'observer son comportement global et ses performances avant l'implantation en cas réels. Les simulations permettent aussi de mieux appréhender les performances de l'algorithme simulé et éventuellement de proposer des optimisations. Dans [Rak94], l'auteur précise qu'il existe deux types de simulations :

– La simulation de systèmes continus : un système est modélisé sous forme d'équations différentielles qui régissent l'évolution du système. Nous trouvons dans ce cas, les simulations des marchés économiques, les simulations écologiques,etc,
– La simulation de systèmes à événements discrets : la modélisation de tels systèmes correspond à des règles de succession d'événements. L'évolution du système dépend alors des événements déjà survenus et de l'ordre temporel de ceux-ci.

Dans notre cas, il s'agit alors de simulations à événements discrets. Le recours à des simulations est nécessaire lorsque :

1. Il n'est pas possible de prédire le comportement qu'adopte le système,
2. Le coût de test du système en conditions réelles est très prohibitif. On procède alors à des simulations pour effectuer les ajustements à priori.

Dans cette section, nous allons présenté différents résultats de simulations concernant le nombre de transitions (définition 3.9) pour la stabilisation et les mécanismes de gestion de pannes présentés dans la section 3.6.3 afin d'observer le comportement en fonction de plusieurs paramètres comme le nombre de nœuds ou le degré moyen (nombre moyen de voisin) dans le graphe.

4. Distributed Algorithm Simulator Of Reims University

Simulation sans modèle de pannes

Les résultats donnés dans cette section ont été obtenus par simulations en utilisant un modèle sans faute. Nous avons obtenues ces résultats en effectuant des séries suffisamment importantes. Toutes nos simulations sont effectuées sur des graphes aléatoires, avec des états de nœuds aléatoires. Nous fournissons ici des valeurs moyennes.
Dans un premier temps, nous avons fixé le degré moyen et nous faisons varier le nombre de nœuds. Dans un deuxième temps, nous avons fixé le nombre de nœuds et nous faisons varier le degré moyen.

Périodes de stabilisation en fonction du nombre de nœuds

Nous avons fixé le degré moyen à 10. Ainsi nous augmentons progressivement le nombre de nœuds de 100 à 1000 par pas de 100, et nous obtenons les résultats présentés dans la figure 3.12. Elle représente le nombre de transitions (définition 3.9) en fonction du nombre de nœuds. Dans la section 3.7, nous avons montré formellement que dans le pire des cas, nous avons un temps de stabilisation de n+2 transitions, avec n le nombre de nœuds du graphe. En pratique, il est très rare de se trouver dans le pire des cas. On remarquera que malgré l'augmentation de nombre de nœuds dans nos graphes aléatoires utilisés, le nombre de transitions pour la stabilisation varie très légèrement, tout en restant très largement en dessous de n+2 transitions. Malgré le fait de faire plusieurs simulations et à chaque fois sur des graphes aléatoires. Ainsi, ces résultats montrent que notre algorithme est purement local et ne nécessite qu'une information sur le voisinage obtenue grâce aux messages *hello*. On peut voir sur la figure 3.12 que lorsque le nombre de nœuds augmente de 500 à 1000, le nombre de transitions se stabilise. Ainsi, notre solution supporte le passage à l'échelle du réseau.

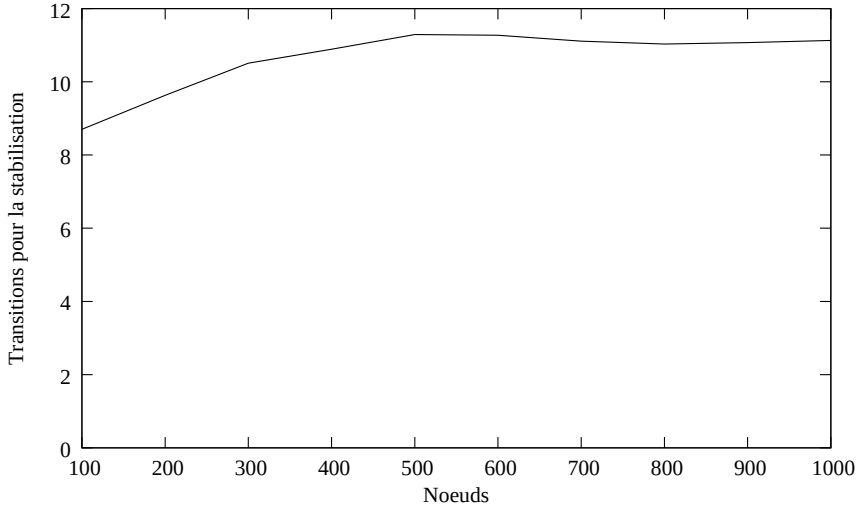


FIGURE 3.12 – Nombre de transitions en fonction du nombre de nœuds.

Périodes de stabilisation en fonction du degré moyen

Le nombre de nœuds est fixé à 100. Nous faisons varier progressivement le degré moyen jusqu'à 50 par pas de 5, et nous observons les variations de transitions. L'augmentation du

degré se traduit par une augmentation du nombre de voisins. Cela signifie que le nombre d'interactions augmente, et donc, chaque nœud doit attendre un nombre important de nœuds pour se fixer. Malgré l'augmentation du degré, le nombre de transitions pour la stabilisation semble augmenter que très légèrement. Nous remarquons aussi sur la figure 3.13 que lorsque le degré augmente de 30 à 50, le nombre de transitions baisse. Cela peut s'expliquer par le fait qu'on a un diamètre assez faible et on s'approche d'un graphe complet. Dans un graphe complet, en 2 transitions tous les nœuds seront fixés. A la configuration C_0, le nœud ayant la plus grande identité est fixé et à la configuration C_1, tous ses voisins sont fixés.

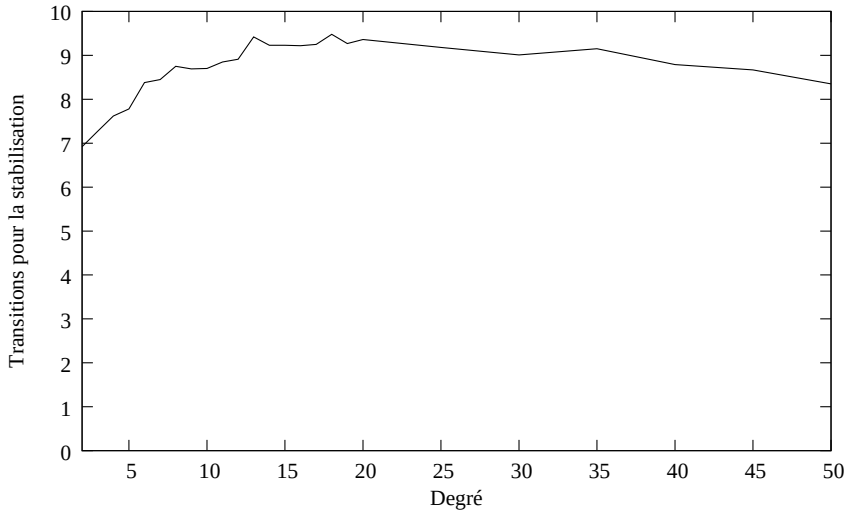


FIGURE 3.13 – Nombre de transitions en fonction du degré moyen.

Périodes de stabilisation en fonction du nombre de nœuds et du degré moyen

La figure 3.14 représente le nombre de transitions en fonction du nombre de nœuds et du degré moyen. Nous remarquons que si l'on fait varier le nombre de nœuds et le degré en même temps, le nombre de transitions varie sensiblement de 7 à un maximum de 10 transitions. Donc la variation du nombre de nœuds et du degré en même temps semble ne pas avoir une grande influence sur le temps de convergence. Ce qui montre que notre algorithme est auto-stabilisant, quel que soit l'état initial des nœuds et sans occurrence d'une faute, le réseau se stabilise au bout d'un nombre fini d'étapes (7 à 10 transitions). C'est également une autre façon d'observer que la complexité de n+2 établie de manière formelle dans la section 3.7 est un cas très rare que l'on rencontre difficilement dans la pratique.

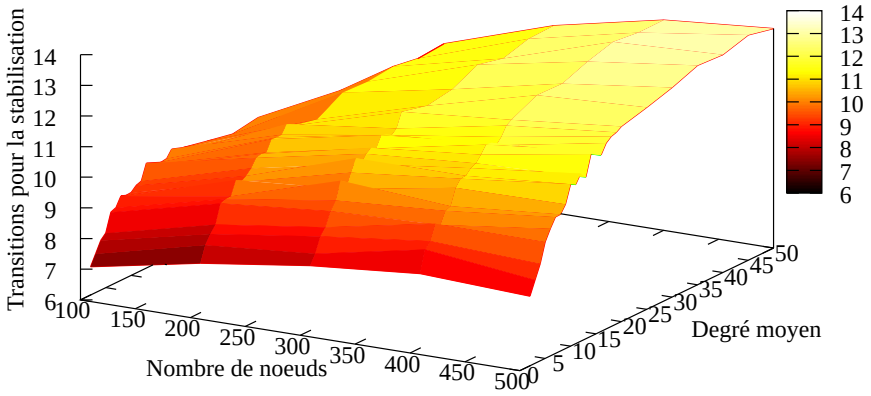


FIGURE 3.14 – Nombre de transitions en fonction du nombre de nœuds et du degré moyen.

Nous présentons ici nos résultats expérimentaux sous forme de errorbars (y-errorbars) pour voir en détails la stabilisation en fonction de la variation du degré moyen. La figure 3.15 montre que sur une série de plusieurs simulations pour chaque pas, le nombre de transitions matérialisé par la moyenne géométrique, varie sensiblement de 7 à 9 transitions. Elle montre également le temps de stabilisation des nœuds qui se stabilisent en dernier (symbolisé par le max de y-errorbars). Là aussi, nous observons sur la figure que, indépendamment du degré moyen, le temps de stabilisation des nœuds qui se stabilisent en dernier est compris entre 10 et 15 transitions. Ce qui est nettement inférieur à la complexité formelle de n+2 transitions.

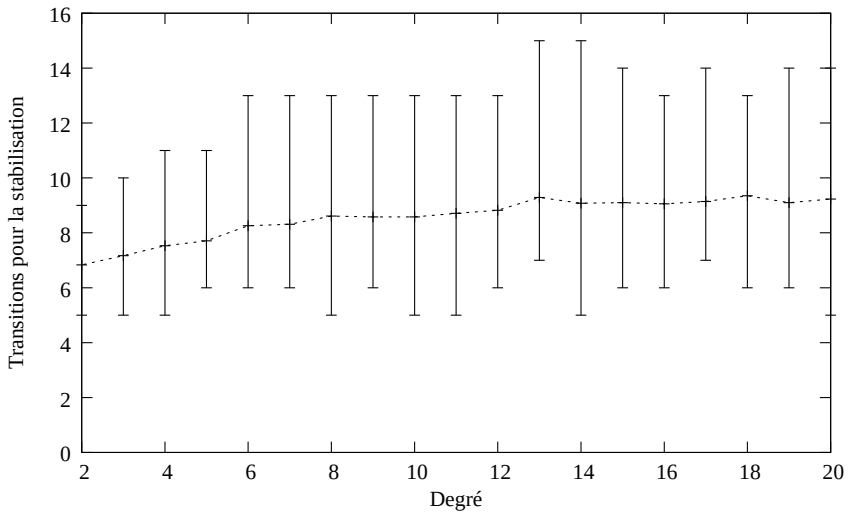


FIGURE 3.15 – Nombre de transitions en fonction du degré moyen.

Après avoir observé expérimentalement que la stabilisation fournie par notre algorithme est indépendante du nombre de nœuds et du degré moyen. Nous allons à présent voir comment notre algorithme assure la répartition des nœuds entre les clusters.

Nombre de clusters en fonction du degré moyen

La figure 3.16 illustre la variation du nombre de clusters en fonction du degré moyen. Nous avons effectué ces simulations en fixant le nombre de nœuds à 100, et nous faisons varier le degré moyen. Nous remarquons que le nombre de clusters décroit logiquement en fonction du degré puisque plus le degré augmente et plus un nœud peut couvrir de nœuds dans son voisinage. Plus le degré augmente, plus les nœuds possédant les identités les plus grandes absorbent plus de nœuds dans leurs clusters ; ce qui implique logiquement une diminution du nombre de clusters. Pour des degrés supérieurs à 30, les résultats sont très proches, et on note plus une différence significative.

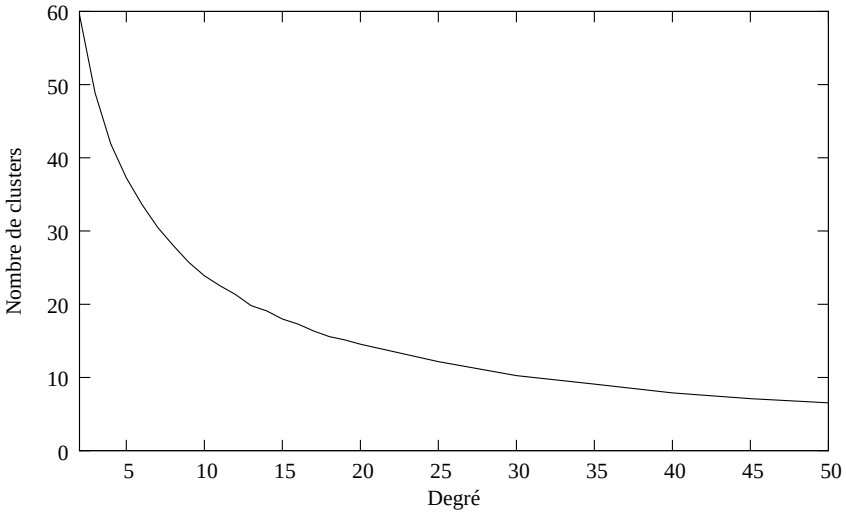


FIGURE 3.16 – Nombre de clusters en fonction du degré moyen.

Répartition des nœuds entre les clusters

La figure 3.17 illustre le nombre de clusters en fonction de leur taille. Nous avons effectué une série de simulations en fixant le nombre de nœuds à 100 et le degré moyen à 10. Nous remarquons que la répartition de nœuds entre les clusters n'est pas homogène. La figure montre aussi que l'on pourrait se trouver dans une situation avec de clusters à un seul nœud. C'est un problème commun aux algorithmes de clustering à 1 saut. Pour cette raison, nous pensons, dans nos futurs travaux, étendre l'algorithme à k sauts. Il faut aussi noté que nous sommes dans un contexte de réseau ad hoc, ce qui suppose que l'on aura une densité importante.

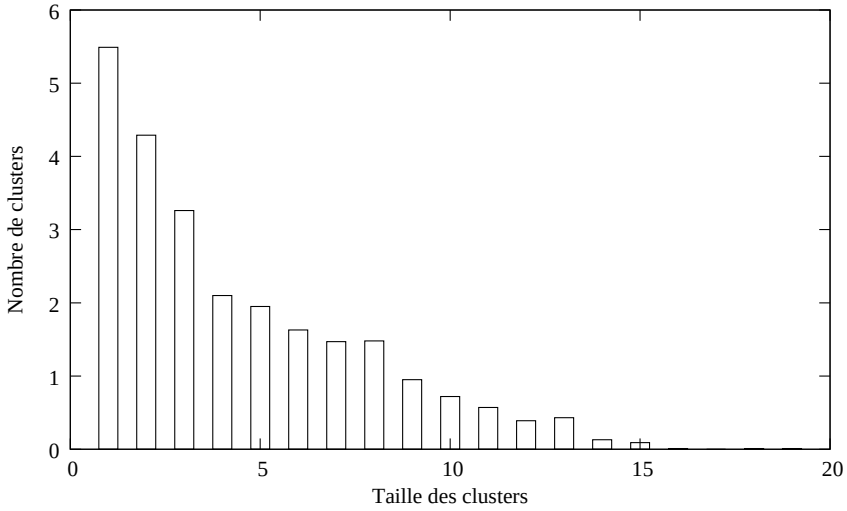


FIGURE 3.17 – Répartition des nœuds entre les clusters.

La figure 3.18 représente la répartition des nœuds entre les clusters en fonction du degré. Nous avons effectué ces simulations en fixant le nombre de nœuds à 100, et nous faisons varier le degré. Là aussi, nous observons sur la figure que le nombre de clusters diminue en fonction du degré moyen. Nous observons également que la taille des clusters ne dépasse pas 10 quelque soit le degré moyen. Ainsi, on a moins de clusters de grande taille.

Ce document doit être joint au dossier transmis
aux rapporteurs désignés par le Conseil National des Universités

DÉCLARATION DE CANDIDATURE À LA QUALIFICATION
AUX FONCTIONS DE MAÎTRE DE CONFÉRENCES,
POUR LA SECTION 27-Informatique
(Campagne 2012)

Je soussigné(e) M.
Nom patronymique : HAGGAR
Nom d'usage ou marital :
Prénom : BACHAR SALIM
Date et lieu de naissance : 15/05/1983 - IRIBA
Nationalité : Hors Europ

Adresse postale et électronique à laquelle seront acheminées toutes les correspondances
2 CHEMIN DE ROULIERS Code postal : 51100 Ville :REIMS Pays : FRANCE Téléphone : 0613916534 Télécopie : Adresse électronique : bachar-salim.haggar@univ-reims.fr

Date de création de la candidature
17/10/2011 à 17:10

Date de dernière modification de la candidature
17/10/2011 à 18:10

Titres universitaires français :
Doctorat

Diplôme au titre duquel la qualification est demandée : Thèse
Titre : Auto-organisation et routage dans les réseaux mobiles ad hoc
Date de soutenance : 30/06/2011
Lieu de la soutenance : UNIVERSITE DE REIMS CHAMPAGNE-ARDENNE
Mention : Très honorable
Directeur : OLIVIER FLAUZAC ET FLORENT NOLOT
Composition du jury : MARC BUI
MICHAEL KRAJECKI
IBRAHIMA NIANG
HERVE GUYENNET
MOHAMED MOSBAH

Simulation avec modèle de pannes

Dans la section 3.6.3, nous proposons de gérer le changement de topologie à l'aide d'un message de type *hello*. Nous allons nous intéresser au changement de topologie dans un cas précis : disparition d'un nœud dans le réseau. Nous considérons un réseau composé de 100 nœuds et nous faisons varier le degré moyen. À la date T_0, tous les nœuds sont stables tandis qu'à la date T_1 ($T_1 > T_0$), nous introduisons un changement de topologie lié à la disparition d'un nœud. Plus précisément, nous faisons une série de plusieurs simulations pour mesurer la disparition de chacun des nœuds. Nous observons sur la figure 3.19 que, le nombre de transitions supplémentaire pour atteindre l'état stable suite à l'occurrence d'une faute est assez faible (2 transitions au maximum jusqu'à des degrés de 20) : cela montre que notre algorithme s'adapte bien au changement de topologie lié à la disparition d'un nœud. Les résultats montrent clairement que le nombre de transitions est largement inférieur à celle d'un réseau non structuré. Les courbes précédentes montrent que, pour des réseaux non structurés, le nombre de transitions varie sensiblement de 7 à 11 transitions. Nous pouvons observer sur la figure qu'en deux transitions supplémentaires le réseau redevient stable. En effet, l'impact d'une faute est purement local, ne concerne que le cluster où a disparu le nœud, et n'influe en rien la stabilité des autres clusters.

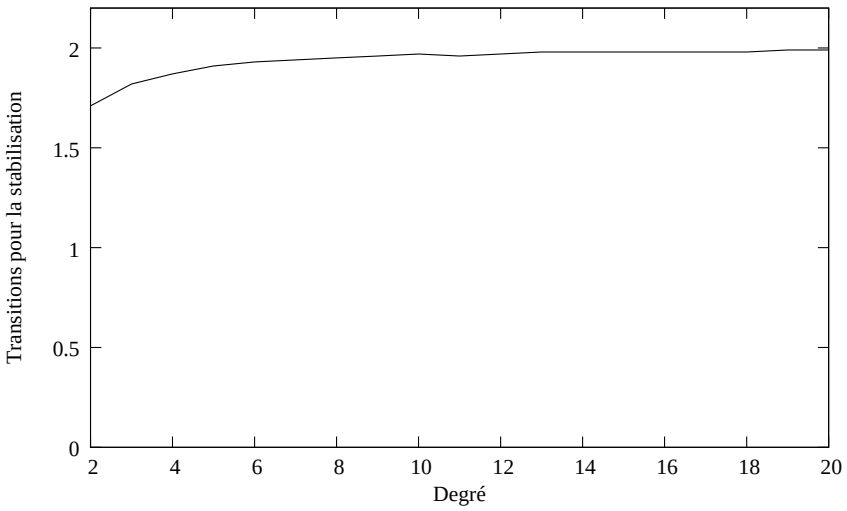


FIGURE 3.19 – Disparition d'un nœud.

3.9 Conclusion

Nous avons proposé dans ce chapitre un nouvel algorithme déterministe pour partitionner un réseau quelconque en clusters. L'objectif principal est d'introduire un premier niveau de hiérarchie dans le réseau. L'algorithme que nous développons à cet effet est un algorithme auto-stabilisant qui n'est basé que sur des connaissances locales : chaque nœud ne connaît que la liste de ses voisins. Cela permet ainsi de réduire considérablement la consommation de bande passante en évitant de faire des broadcast ou multicast. L'algorithme garantit aussi qu'à partir d'une configuration quelconque, le réseau sera entièrement organisé en cluster en au plus n+2 transitions où n est le nombre de nœuds du réseau. Chaque nœud n'appartiendra qu'à un seul cluster en au plus n+2 transitions. Contrairement à la plupart des autres algorithmes de clustering, nous utiliserons qu'un unique message pour découvrir le voisinage d'un nœud et faire le clustering. La force de cet algorithme réside d'une part s'adapte aux modifications topologiques et d'autre part réduit le trafic réseau en combinant la découverte réseau, la construction et la maintenance des clusters en une seule phase.

Nous avons également démontré formellement que l'algorithme est auto-stabilisant. Ainsi à partir d'une configuration quelconque, même si un changement de topologie survient, par exemple si un nouveau nœud apparait, ou un nœud disparait, l'algorithme converge vers un état légal. Donc l'auto-stabilisation est une caractéristique importante de notre algorithme.
Nous avons donc proposé un algorithme qui permet de hiérarchiser le réseau en clusters. Comme nous avons proposé une solution permettant de hiérarchiser le réseau , il est donc maintenant tout à fait logique d'exploiter cette solution. Ainsi nous allons dans un premier temps utilisé cette structure pour pouvoir faire de la diffusion d'infirmations dans le réseau (chapitre 4) et ensuite du routage (chapitre 5).
Les travaux présentés dans ce chapitre ont fait l'objet de deux publications en 2009 [FHN09, Hag09].

Chapitre 4

Diffusion

Résumé : *Nous présentons d'abord dans ce chapitre les différentes solutions de diffusion d'informations dans le réseau. Nous présentons ensuite une solution pour la construction de l'arbre couvrant inter-clusters. Elle est basée sur des connaissances locales et complètement distribuée. Chaque nœud exécute l'algorithme de façon distribuée et choisit le cluster possédant le plus grand identifiant connu comme père. Elle s'adapte aux modifications topologiques et ne nécessite que peu d'échanges de messages. Nous proposons d'utiliser cette solution pour diffuser de l'information dans le réseau. En effet, l'arbre couvrant inter-clusters apporte un avantage dans la diffusion d'informations, car il permet de réduire le nombre de nœuds qui sont chargés de relayer les messages et d'atteindre la totalité des nœuds du réseau. Cependant, la diffusion peut être divisée en deux étapes : une diffusion locale qui consiste à acheminer le message à tous les nœuds au sein du cluster, et la diffusion globale qui consiste à envoyer le message à tous les nœuds du réseau.*

4.1 Introduction

Dans le chapitre précédent, nous avons proposé une solution permettant d'organiser le réseau en clusters. Comme nous l'avons vu, une telle organisation présente de nombreux avantages. Cependant, elle peut être utilisée pour faire de la diffusion d'informations dans le réseau. Un des problèmes étudiés par la communauté des chercheurs sur les réseaux ad hoc est la diffusion d'informations dans le réseau. Le but principal de ces travaux est la réduction du nombre de réémissions lors de la diffusion d'un message à l'ensemble du réseau.
La diffusion est l'opération qui consiste à transmettre un message à l'ensemble des nœuds du réseau. Cette technique est nécessaire dans de nombreux cas, notamment lors de la recherche d'une route dans le réseau. Dans ce cas un nœud source S initie une découverte de route en diffusant un message qui contient l'identité du nœud destinataire D. Quand le message atteint sa destination avec succès, le nœud destinataire D répond au nœud source S en utilisant un message court contenant la séquence des nœuds à travers laquelle il peut être atteint. Le nœud source S peut alors utiliser cette route pour envoyer le message au nœud destinataire D. Cette technique est notamment utilisée par les protocoles réactifs comme DSR[JHM07, JMJ01].

La façon la plus simple pour mettre en place une technique de diffusion est d'utiliser le protocole d'inondation *(blind flooding [ASTC96])*. Son fonctionnement est simple, chaque nœud

qui reçoit un message de diffusion pour la première fois le transmet à tous ses voisins directs. La seule optimisation appliquée à cette solution est que les nœuds se souviennent des messages reçus par diffusion et ne retransmettent pas les copies du même message. Ce mécanisme nécessite l'utilisation de numéro de séquence comme dans [JHM07] [PE99, PBRD03]. Lors de l'envoi d'un message de diffusion, le nœud à l'origine de la diffusion génère un numéro de séquence et l'ajoute au message de diffusion. Ce numéro de séquence avec l'identifiant du nœud source et de la destination identifie de façon unique un message dans le réseau. Après traitement d'un message de diffusion, chaque nœud stocke temporairement cette information. Ainsi, chaque nœud est en mesure d'évaluer la "fraîcheur" d'un message de diffusion par rapport à un autre afin de détecter les copies du même message.

Cette méthode est déjà utilisée dans les réseaux filaires (dans le protocole de transport TCP [1] par exemple) pour permettre la fragmentation des paquets, le numéro de séquence utilise comme unité le nombre d'octets transmis.

Malgré sa simplicité, *blind flooding* entraîne une charge importante du réseau et génère des problèmes tels que le gaspillage de bande passante, collisions, ...

Comme nous l'avons dit précédemment, pour toute conception d'une solution destinée aux réseaux ad hoc nécessite d'abord une organisation du réseau. Ainsi la diffusion d'information dans le réseau ad hoc nécessite à la fois l'organisation du réseau et un algorithme de diffusion. Nous proposons un algorithme de construction d'arbre couvrant basé sur notre structure en clusters. Notre algorithme est un algorithme de réduction de graphes diminuant le nombre de liens de manière à éviter la formation de boucles tout en gardant la connectivité complète du réseau. Ainsi avec une telle solution, nous pouvons réduire les retransmissions redondantes et économiser la consommation de bande passante. La plupart des algorithmes existants se basent sur la connaissance complète de la topologie du réseau pour permettre la construction d'arbres couvrants. Nous pensons que ce genre d'algorithmes ne sont pas adaptés aux réseaux ad hoc vu les contraintes assez fortes de ces réseaux. Comme nous l'avons fait dans la section 3.3, nous utiliserons un algorithme localisé distribué pour la construction de notre structure. Ainsi aucune connaissance globale n'est nécessaire pour parvenir à faire un arbre couvrant de clusters.

Nous allons dans un premier temps présenter les algorithmes de diffusion pour les réseaux ad hoc dans la littérature. La section 4.3 décrira notre solution.

4.2 Les techniques de diffusion des réseaux ad hoc

Un algorithme de diffusion est dit fiable s'il permet de joindre tous les nœuds du réseau. D'un point de vue local, chaque nœud doit alors garantir que l'ensemble de son voisinage est contacté par le message de diffusion. En plus il doit aussi limiter l'utilisation de la bande passante et la consommation d'énergie. De ce fait, il doit minimiser le nombre de messages retransmis et la réception des copies du même message.
Plusieurs algorithmes ont été proposé dans la littérature. La plupart de ces algorithmes con-

1. Transport Control Protocol

sidèrent que la couche MAC [2] est idéale. Cette hypothèse est très important parce qu'elle influe directement sur la fiabilité du protocole. En considérant que la couche MAC est idéale, aucune transmission n'est perturbée par une autre, et aucun message n'est perdu par la suite de collision.
Dans cette section, nous allons faire un état de l'art complet sur les travaux existants.

L'algorithme de diffusion le plus trivial pour diffuser un message dans le réseau est l'inondation aveugle (blind flooding). Son fonctionnement est simple, chaque nœud qui reçoit un message de diffusion pour la première fois le transmet à tous ses voisins. Le défaut de cet algorithme est lorsque deux nœuds très proches se décident à réémettre le même message, une grande partie des voisins de ces deux nœuds vont recevoir exactement le même message. De plus, tous les voisins vont essayer de réémettre le message. Cette utilisation de doublons est pourtant inutile et impose une charge énorme au réseau.
Ce problème est connu sous le nom de tempête de diffusion *(Broadcast Storm Problem)*. Il a été étudié en détail par les auteurs dans [NTCS99]. Le but des auteurs est de réduire le *Broadcast Storm Problem*. Pour cela, ils proposent cinq méthodes :

Porbabiliste (Probabilistic Scheme)

Lorsqu'un nœud reçoit un message pour la première fois, il le retransmet avec une probabilité P, fixée à l'avance. Chaque nœud possède les mêmes chances de réémettre le message. Quand $P = 1$, cette solution est équivalente à la diffusion aveugle.

Schéma basé sur le comptage (Counter-Based Scheme)

Chaque nœud attend un certain temps entre la réception et la réémission. Durant cette période, s'il reçoit le message de diffusion, il diffère à nouveau sa transmission. Chaque nœud utilise un compteur c pour garder une trace du nombre de fois ou il a reçu le message de diffusion. Le compteur c est initialisé à 1 et à chaque réception du message, il sera incrémenté de 1. Un nœud décide de transmettre le message que si $c \leq C$. Dans le cas contraire il ignore cette transmission. Donc un nœud ne transmet pas le message s'il a déjà plus de C fois. Dans ce cas il y a une chance que les autres nœuds reçoivent le même message. Cette heuristique donnent des bons résultats en termes d'accessibilité en fixant la valeur du C supérieure ou égale à 3. Par contre, le Saved ReBroadcasts (nombre de nœuds recevant le message de diffusion et ne le réémettent pas) est fortement dépendant de la densité du réseau.

Schéma basé sur la distance (Distance-Based Scheme)

Dans ce schéma, les nœuds utilisent la distance relative comme critère pour la réémission des messages. Un nœud ne réémet pas un message, s'il reçoit d'un nœud situé à une distance inférieure à d. Comme dans l'approche précédente, chaque nœud attend un certain temps entre la réémission et la réception du message, pour laisser une chance aux autres nœuds voisins d'envoyer leurs messages. Si un de ces messages arrive à une distance inférieure à d alors il annule son émission. Dans le cas contraire, à l'issue du temps d'attente, le nœud réémet le

2. Media Access Control

message de diffusion. Les auteurs utilisent la puissance du signal pour estimer la distance d. Cette méthode donne aussi de bons résultats en termes d'accessibilité. Par contre elle donne de très mauvais résultats en termes de Saved ReBroadcasts. Cela est dû du fait que même si un nœud a entendu plusieurs fois le message de diffusion et que les nœuds qui ont émis le message sont situés à une distance supérieure à d, il peut quand même le réémettre.

Schéma basé sur la localisation (Location-Based Scheme)

Dans cette approche, chaque nœud doit être en mesure de calculer sa position afin d'estimer sa couverture supplémentaire de façon précise. Les nœuds obtiennent les informations des localisations grâce au GPS [3]. Chaque nœud ajoute les informations concernant sa localisation dans l'entête du message qu'il envoie ou réémet. Quand un nœud reçoit un message, la première chose qu'il doit faire est de noter les informations de localisation du nœud émetteur et ensuite calculer la zone de couverture supplémentaire pour la réémission. Si la zone de couverture pour la réémission est inférieure à un seuil A alors le message sera ignoré. Sinon le message sera réémis. Comme dans les approches précédentes, un délai d'attente entre la réception et l'émission est utilisé afin d'optimiser le nombre de voisins qui émet leur message. L'inconvénient majeur de cette approche est le coût induit par le calcul de la zone de couverture supplémentaire, qui est un calcul du nombre d'intersections entre plusieurs cercles. Cette approche obtient des meilleurs résultats dans tous les aspects par rapports aux autres approches. Cela est dû au fait que les nœuds utilisent les informations exactes pour calculer la zone de couverture supplémentaire. Elle fournie une accessibilité proche de 100% tout en offrant une meilleure économie d'énergie. Le temps de latence est également meilleur que les autres approches.

Schéma basé sur les cluster (Cluster-Based Scheme)

Nous avons décrit en détail la formation des clusters dans le chapitre 3. Dans cet algorithme les nœuds utilisent l'algorithme de LCA (Link Cluster Architecture) ou HCC (hight Conectivity Clustering) pour la formation des clusters. L'idée de cette approche est qu'un nœud qui a le plus petit identifiant ou plus grand degré de connectivité devient clusterhead. Ses voisins à 1 saut s'attachent à lui et deviennent nœuds membres. Si un nœud se trouve être entouré par deux ou plusieurs clusterheads, il devient un gateway. Lors de la diffusion, seuls les clusterheads et les gateways transmettent le message de diffusion. Les nœuds membres ne diffusent pas les messages de diffusion. Cette approche donne des meilleurs résultats en termes de Saved ReBroadcasts. Malheureusement, elle offre une accessibilité moyenne en cas de densité faible. Cela est due à la réduction du nombre de nœuds chargés de relayer le message de diffusion.

4.2.1 Les algorithmes dépendants de la source

Pour que la diffusion soit efficace, il faut garantir que tous les nœuds à deux sauts de la source reçoivent le message. Si cette technique est utilisée correctement par chaque nœud alors tous les nœuds connexes du réseau pourront être joints. L'objectif de ces algorithmes est de minimiser le nombre de nœuds qui réémettent le message de diffusion. Pour cela un sous-ensemble

3. Global Positioning System

des voisins à 1 saut peut suffire pour joindre l'ensemble des voisins à deux sauts. Lorsque ce sous-ensemble est choisi, chaque nœud émet son message avec la liste des voisins qui devront réémettre le message de nouveau. Ce sous-ensemble est appelé des nœuds relais. Le problème est de trouver ce sous-ensemble des nœuds relais. Les auteurs ont montré dans [QVL02] que trouver ce sous-ensemble de nœuds relais est un problème NP-complet. Ainsi pour déterminer ce sous-ensemble de façon optimale, elle nécessite l'utilisation d'une approche heuristique.

Dans [QVL02], les auteurs ont proposé une méthode de diffusion par relais multi-points (Multipoint Relaying ou MPR). Dans ce protocole, chaque nœud choisi un sous-ensemble de nœuds voisins pour joindre l'ensemble des nœuds à deux sauts. Lors de la diffusion, chaque nœud ajoute à son message la liste des voisins relais qui devront réémettre le message. Pour calculer un sous-ensemble des nœuds relais, il nécessite la connaissance de la topologie à deux sauts. Ainsi chaque nœud diffuse régulièrement la liste de ses voisins en utilisant le message *hello*. Lors de la diffusion, un nœud diffuse un message que seulement s'il le reçoit pour la première fois et il a été choisi comme un nœud relais. Après chaque changement de la topologie locale, chaque nœud calcule à nouveau un ensemble des voisins relais pouvant joindre l'ensemble des voisins à deux sauts. Plus le sous-ensemble est petit, plus la diffusion sera efficace. Trouver un sous-ensemble optimal est un problème NP-complet Comme le montre les auteurs. Les auteurs ont proposé une heuristique simple. Soit x un nœud, $N(x)$ est l'ensemble des voisins du nœud x et $N^2(x)$ l'ensemble des voisins à deux sauts de x. Le sous-ensemble des voisins relais du nœud x est noté $MPR(x)$.

Algorithme 5 Algorithme de sélection d'un sous-ensemble relais sur un nœud x

Étape 1
Placer dans $MPR(x)$ les nœuds de $N(x)$ qui sont les seuls voisins des nœuds isolés de $N^2(x)$
Étape 2
Tant qu'il existe des nœuds $v \in N^2(x)$ qui ne sont pas couverts, placer dans $MPR(x)$ un nœud $u \in N(x)$ qui couvre plus des nœuds non couverts.

Afin de mieux comprendre le fonctionnement de cet algorithme, exécutons sur le nœud 1 de la figure 4.1. Les nœuds isolés de 1 sont $\{4, 7, 10, 13\}$. Pour la première étape de l'algorithme, les nœuds 2 et 3 seront sélectionnés car ce sont les seuls nœuds qui couvrent les nœuds isolés. Donc à l'étape 1, nous obtenons MPR(1)= $\{2, 3\}$. Maintenant le nœud 1 passe à la seconde étape de l'algorithme et sélectionne le nœud 5 car le nœud 5 possède le plus grand nombre des voisins à deux sauts de 1, non couverts par l'étape précédente. Ainsi tous les nœuds sont couverts par le sous-ensemble choisi comme relais, l'algorithme s'arrête. Donc MPR(1)= $\{2, 3, 5\}$. Comme on peut le constater, cette technique réduit de façon considérable *l'overhead* par rapport à l'algorithme de *blind flooding*, où chaque nœud qui reçoit un message pour la première fois le transmet à tous ses voisins.

4.2.2 Diffusion basée sur les clusters

L'idée sous-jacente des techniques cluster-based est d'introduire une hiérarchie dans le réseau, en créant des clusters. Le but est toujours le même, c'est d'éliminer les messages redondants et de maximiser le Saved-Broadcasts lors d'un processus de diffusion. Plusieurs

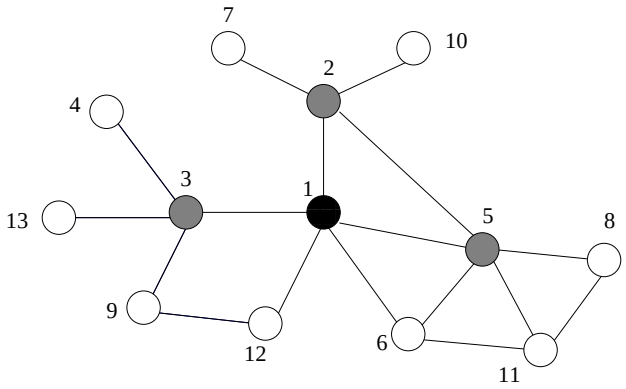


FIGURE 4.1 – Exemple de sélection d'un sous-ensemble des voisins relais avec MPR

.

algorithmes de construction des clusters ont été proposés dans la littérature comme par exemple l'algorithme de LCA (Link Cluster Architecture) ou HCC (hight Conectivity Clustering). Nous avons détaillé ces algorithmes dans le chapitre 3.3. Dans le processus de diffusion seuls les clusterheads et les gateways diffusent le message.

Dans [MF05], les auteurs ont proposé un algorithme de diffusion basé sur des clusters. Dans le processus de formation des clusters, ils ont introduit une nouvelle métrique appelé densité. La densité d'un nœud $u \in V$ est :

$$\rho(u) = \frac{|\{e = (v, w) \in E | w \in \{u\} \cup \Gamma_1(u) \; et \; v \in \Gamma_1(u)\}|}{\delta(u)} \tag{4.1}$$

Pour calculer la densité, l'algorithme nécessite la connaissance de la topologie à deux sauts. Ainsi chaque nœud calcul individuellement sa densité et diffuse à tous ses voisins à 1 saut. Le nœud ayant la plus grande densité dans son voisinage s'élit clusterhead et ses voisins deviennent nœuds membre. Après la formation des clusters, ils proposent d'appliquer un algorithme de diffusion. Cet algorithme construit des clusters à k sauts.
Ainsi dans le processus de diffusion, trois types de diffusions sont utilisées :

- Diffusion dans le voisinage : elle consiste à envoyer un message à ses voisins (comme les messages hello utilisés pour le contrôle de la topologie),
- Diffusion localisée : diffusion dans un cluster uniquement,
- Diffusion globale : dans ce cas le message sera diffusé dans tout le réseau.

Afin de distinguer ces trois types de diffusions, il nécessite une indication dans le message de diffusion. Lors de la diffusion dans un cluster, tous les nœuds internes appartenant à ce cluster réémettent le message. Par contre, quand le message est diffusé dans tout le réseau, tous les nœuds internes du réseau et les gateways réémettent le message. Les gateways réémettent le message de diffusion sous certaines conditions. Un gateway $GW(C(u), C(v))$ réémet un message seulement s'il arrive de son propre cluster $C(u)$. Un gateway miroir $GWm(C(u), C(v))$ ne réémet le message que s'il arrive du cluster $C(u)$ pour lequel il est miroir. Cependant, un gateway miroir $GWm(C(u), C(v))$ réémet le message provenant du cluster $GWm(C(u), C(v))$, peu importe le nœud qui lui envoie le message. En plus lors de la diffusion dans tout le réseau, tous les gateways ne sont pas nécessairement utilisés. Par exemple quand il y a plusieurs gateways vers

deux clusters, seul un seul gateway sera utilisé.

4.2.3 Diffusion basée sur le mécanisme d'élimination des voisins (NES)

Dans le but d'éliminer les messages redondants, les auteurs ont proposé dans [IMZ02] un protocole simple basé sur un mécanisme d'élimination des voisins ou *neighbor elimination scheme (NES)*. Il permet d'éliminer les messages redondants par une simple écoute des transmissions du voisinage. Ce mécanisme utilise la politique de *"Wait and See"*, où un u ne réémet pas immédiatement un message mais il attend un certain temps pendant lequel il va écouter les transmissions de ses voisins à 1 saut. Le temps d'attente peut être choisi de manière aléatoire. A la fin de ce temps, si certains de ses voisins n'ont pas toujours reçu le message, alors u diffusera le message. Au contraire, si tous ses voisins ont déjà reçu le message alors la réémission n'est pas nécessaire et elle sera ignorée. Il est aussi possible que certains voisins de u aient reçu le message par l'intermédiaire de ses voisins à 2 sauts sans que u ne le sache. Dans ce cas, u réémettra le message bien que cette transmission est inutile. Donc ce procédé n'est pas parfait.

4.2.4 Diffusion basée sur les ensembles dominants connexes

Un ensemble dominant connecté ou *connected dominating set (CDS)* est un ensemble de nœuds tel que tout nœud du réseau est au moins voisin d'au moins un nœud du CDS et tel que le CDS forme une structure connexe. Soit un ensemble dominant V_d, avec $V_d \in V$. Plus précisément, V_d est dit dominant si et seulement si :

$$\forall\ u \in V,\ u \in V_d \vee (\exists w \in V_d\ /\ u \in N_w) \tag{4.2}$$

Les nœuds appartenant à V_d sont appelés dominants et les autres nœuds étant des dominés. Cette technique permet de faire une diffusion complète dans tout le réseau en utilisant uniquement les nœuds de V_d, car les nœuds de V_d couvrent l'ensemble des nœuds du réseau comme montre la figure 4.2. La diffusion dans le réseau est alors simple, seuls les nœuds de V_d réémettent le message de diffusion.

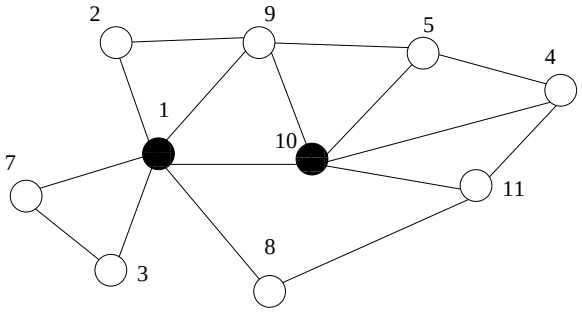


FIGURE 4.2 – Ensemble dominant $V_d = \{1, 9, 10\}$

.

Dans [WL99], les auteurs ont proposé une méthode simple pour construire un ensemble dominant connexe. Cette construction nécessite la connaissance de la topologie à deux sauts.

Les auteurs introduisent le concept de nœud intermédiaire. Un nœud u est dit intermédiaire si et seulement si au moins deux de ses voisins v et w ne sont pas eux mêmes voisins. Donc le nœud u est un nœud intermédiaire entre v et w.
Deux règles d'élimination sont utilisées par la suite pour réduire l'ensemble dominant constitué de nœuds intermédiaires :

- Règle 1 : un nœud u devient dominé s'il est couvert par un voisin v, c'est à dire $N_u \subset N_v$ et $id_u < id_v$,
- Règle 2 : un nœud u devient dominé s'il est couvert par deux de ses voisins v et w, chaque voisin de u est un voisin de v ou w, $id(u) < id(v)$ et $id(u) < id(w)$,
  Plus formellement : $N_u \subset N_v \cup N_w$, $id(u) < id(v)$ et $id(u) < id(w)$.

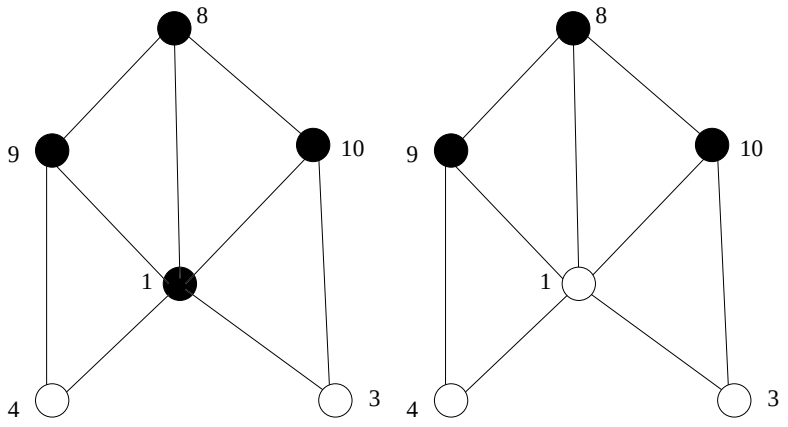
Dans [IMZ02], les auteurs ont proposé une version améliorée de cet algorithme. Dans la version améliorée, l'identifiant du nœud est remplacé par une clef. La clef détermine la priorité d'un nœud d'être dans l'ensemble dominant, elle peut être le degré ou une combinaison des métriques. Ainsi les deux règles précédentes deviennent alors :

- Règle 1 : un nœud u devient dominé s'il est couvert par un voisin v, c'est à dire $N_u \subset N_v$ et $clef(v) > clef(u)$,
- Règle 2 : un nœud u devient dominé s'il est couvert par deux de ses voisins v et w, $N_u \subset N_v \cup N_w$, $clef(v) > clef(u)$ et $clef(w) > clef(u)$.

Comme cette variante privilégie les nœuds possédant le plus fort degré d'être dans l'ensemble dominant, donc elle réduit l'ensemble dominant constitué des nœuds intermédiaires.

Dans [DW03], les auteurs ont proposé une généralisation à k-voisins des règles précédentes. Dans cette règle, un nœud u appartenant à un sous-ensemble V_d doit être retiré de V_d si les trois conditions suivantes sont vérifiées :

- Le sous-ensemble V_d est connecté,
- Tout voisin de u est d'au moins voisin d'un nœud de V_d,
- Tous les nœuds de V_d possède une priorité supérieure au nœud u.



(a) Sous-ensemble V_d de départ (b) Sous-ensemble V_d réduit

FIGURE 4.3 – Application de la Règle généralisée

La figure 4.3 illustre un exemple de cette méthode. Dans la figure 4.3(b), le nœud 1 a été retiré de V_d car ses voisins sont tous couverts par les nœuds de V_d de priorité plus forte élevée. Donc $V_d = \{8, 9, 10\}$.

4.2.5 Diffusion basée sur le contrôle de topologie

Ces protocoles tentent de modifier la topologie du réseau en s'appuyant sur des algorithmes de réduction de graphes afin d'optimiser la diffusion d'informations dans le réseau. Ainsi ils utilisent un sous graphe $G' = (V', E')$ du graphe d'origine $G = (V, E)$ tel que $(E' \subset E)$. Le but est de supprimer certaines arêtes du graphe d'origine afin de modifier localement la topologie de chaque nœud.

Relative Neighborhood Graph (RNG)

Le graphe de voisinage relatif *(ou Relative Neighborhood Graph)* a été présenté par Toussaint dans [Tou80]. Le graphe de voisinage relatif d'un graphe G est noté par $RNG(G) = (V, E_{rng})$ et défini par :

$$E_{rng} = \{(u, v) \in G | \nexists w \in N_u \cap N_v \ d(u, w) < (u, v) \wedge d(v, w) < d(u, v)\} \tag{4.3}$$

Cette condition est illustrée dans la figure 4.4. Nous avons deux cercles de rayon $d(u, v)$ centrés en u et v. L'arrête (u, v) n'appartient pas au graphe RNG car il existe un w dans l'intersection de ces deux cercles. Les deux arêtes valides du RNG sont (u, w) et (v, w). A partir de là, il est possible de donner une définition simplifiée du calcul du graphe RNG : quels que soient les nœuds u et v, il n'existe aucun nœud w dans l'intersection des cercles centrés en u et v et de rayon uv. L'auteur a montré dans [Tou80] que si le graphe G est connecté alors $RNG(G)$ est lui aussi connecté.

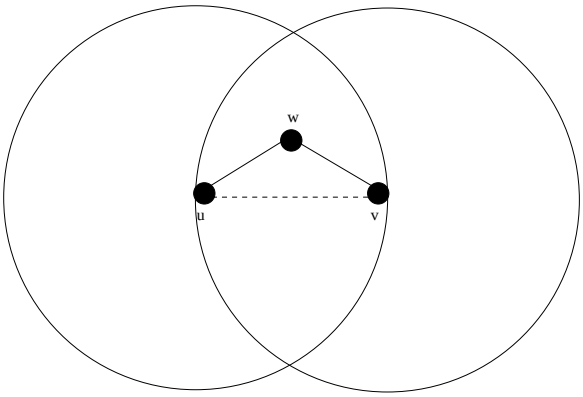


FIGURE 4.4 – Calcul d'un RNG : l'arrête (u, v) n'est pas dans le RNG

Un exemple d'un graphe RNG est donné dans la figure 4.7.

Dans [CIS03], les auteurs proposent la construction d'un RNG afin d'éliminer les messages redondants et ainsi d'optimiser la diffusion dans le réseau. Pour évaluer l'information de distance entre un nœud u et ses voisins, et entre ses voisins, il nécessite l'utilisation d'un GPS . Cependant

(a) Graphe d'un réseau (b) Graphe RNG d'un réseau

FIGURE 4.5 – Exemple d'un graphe RNG

l'utilisation d'un tel équipement est coûteux et en l'absence d'un tel système les nœuds ne sont pas en mesure d'évaluer les distances, c'est pourquoi les auteurs proposent de remplacer la distance par la différence de voisinage. La distance entre deux nœuds u et v est définie par :

$$\nu(u,v) = \frac{|N_u/N_v \cup N_v/N_u|}{|N_u \cup N_v|} \tag{4.4}$$

Chaque nœud u calcule l'ensemble $RRS(u)$, qui est son sous ensemble RNG relais. Le $RRS(u)$ est définie par :

$$\forall u \in V, RRS(u) = \{v \in N_u \mid N_v(rng)/(N_u \cup \{v\}) = \emptyset\} \tag{4.5}$$

En d'autre termes, un nœud v est un relais pour u si et seulement si : v est un voisin de u et v a un voisin RNG qui n'est pas couvert par les transmissions de u. Lors de la diffusion d'un message chaque nœud procède de la manière suivante : A la réception d'un message provenant d'un nœud u, chaque nœud teste s'il appartient ou non au sous ensemble relais de u. Si c'est la cas, le nœud réémet le message, sinon la transmission sera annulée.

La figure 4.6 illustre un exemple de cette méthode. Dans cette figure, si le nœud a diffuse un message, le nœud b, c, d et e ne vont pas réémettre le message. Par contre, les nœuds f, g, h et i vont réémettre le message car chacun possède un voisin RNG n'appartenant pas au voisinage de a.

Local Minimum Spanning Tree (LMST)

Les auteurs ont proposé dans [LHS05] un algorithme appelé Local Minimum Spanning Tree (LMST). Cet algorithme permet de construire un sous-graphe pour réduire le nombre de liens. Cet algorithme fonctionne de la manière suivante : chaque nœud u construit le MST de son voisinage noté $MST(N(u))$. Deux arêtes $(u,v) \in V$ sont conservées dans le graphe LMST de G si et seulement si u est un voisin de v dans le $MST(N(v))$ et v est un voisin de u dans le $MST(N(u))$. Ainsi le sous-graphe obtenu aura les mêmes nombre de nœuds et éventuellement un nombre réduit de liens. Donc $LMST(G) = (V, E_{LMST})$ est le sous-graphe LMST du graphe $G = (E, V)$.
La construction d'une telle structure nécessite la connaissance de la topologie à deux sauts, puisqu'il est nécessaire pour un nœud de connaître ses voisins et les liens entre ses voisins. Cette

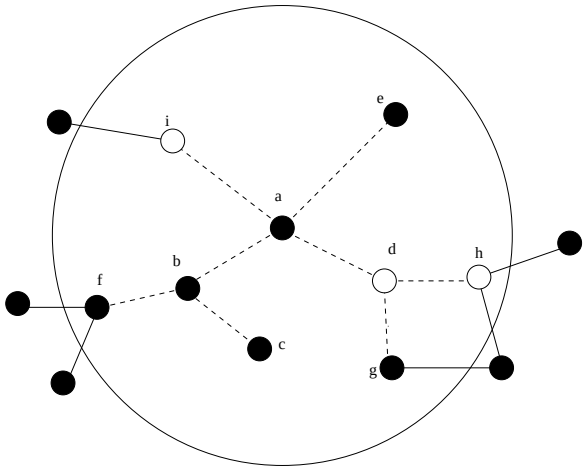
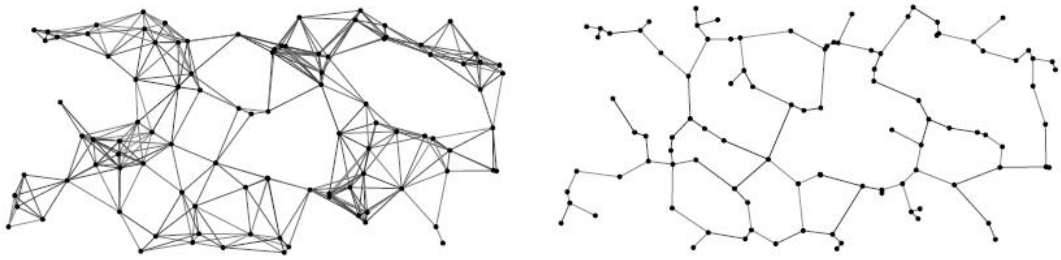


FIGURE 4.6 – Diffusion avec *RNG*

connaissance nécessite l'utilisation d'un équipement spécifique comme le GPS. Les auteurs ont également montré que si le graphe G est connexe alors le LMST obtenu préserve la connexité.



(a) Graphe d'un réseau (b) LMST

FIGURE 4.7 – Exemple d'un graphe et son LMST

Dans [Lav00, LL91, BLB95, DJPV98], les auteurs ont proposé des algorithmes de construction d'arbre couvrant. Pour plus de détails le lecteur intéressé pourra consulter ces références.

4.3 Notre solution

Comme nous l'avons vu précédemment, il existe dans la littérature plusieurs protocoles de diffusion. La plupart d'entre eux nécessitent une connaissance au delà de deux sauts pour pouvoir faire de la diffusion dans le réseau et ne s'adaptent pas aux changements topologiques.

C'est pourquoi, nous proposons un algorithme de construction d'arbre basé sur une structure en clusters. Cet algorithme est basé sur des connaissances locales. Avec notre solution de clustering, la construction de l'arbre se fait sans surcoût en utilisant un unique message *hello*. Notre objectif est à la fois d'optimiser les messages redondants et réduire le nombre de nœuds qui relaient le message. Cependant, notre protocole permet de garantir l'unicité du chemin entre tous les clusters du réseau.

Nous allons dans un premier temps décrire le fonctionnement général de notre solution. Puis nous introduisons les notations utilisées dans ce chapitre. Ensuite, nous présenterons notre algorithme de construction d'arbre couvrant. Enfin, nous présenterons la diffusion d'informations dans le réseau qui exploite cet arbre couvrant.

4.3.1 Description générale

Nous proposons de construire un arbre de clusters enraciné sur le cluster ayant l'identité la plus grande. Chaque sommet de l'arbre correspond à un cluster et une arrête existe entre deux clusters si et seulement si deux nœuds de ces clusters sont voisins. Cet algorithme est également basé sur des connaissances locales et ne nécessite que des interactions locales. Pour cela, nous modifions la structure du message *hello* utilisé dans l'algorithme présenté dans la section 3.3, et nous ajoutons deux champs supplémentaires : les identités des clusters voisins, l'identité de cluster ayant la plus grande identité détectée. Ces informations permettent de construire l'arbre de clusters de façon distribuée sans pouvoir faire de la diffusion dans le réseau.

Ainsi, l'algorithme construit un arbre couvrant inter-clusters de la façon suivant : chaque nœud u exécute l'algorithme de façon distribuée et choisit le cluster ayant la plus grande identité qu'il a détecté comme père (il est aussi la racine). Ensuite, u propage cette information (id_u, et l'id du père de u) en l'incluant dans les messages *hello* envoyés à ses clusters voisins. À la réception du message de u par un nœud v, si l'identité maximale que v a détecté auparavant est plus petite que celle de u, v choisit u comme père et propage cette information (id_v, id_u et l'id du père de u) à ses clusters voisins. Ce processus se répète jusqu'à ce que l'arbre soit construit et que le cluster ayant l'identité la plus grande identité soit choisi comme racine de l'arbre. À la fin du processus, chaque cluster connaît la racine de l'arbre et la liste de clusters permettant de l'atteindre. Chaque cluster connaît également l'ensemble de ses clusters fils et de son père. Le choix du cluster père est basé sur la distance entre lui et le cluster ayant la plus grande identité détectée. Donc, chaque cluster choisit un cluster voisin comme père si celui-ci est plus près de la racine détectée. Le fait que chaque clusterhead maintienne la liste de clusters permettant d'atteindre la racine de l'arbre, permet à l'algorithme de s'adapter aux changements topologiques.

La figure 4.8 représente l'arbre couvrant de clusters obtenu par notre algorithme. Nous remarquons que l'arbre est enraciné sur le cluster ayant la plus grande identité (cluster 15).

4.3.2 Notations

Nous introduisons ici des notations supplémentaires que nous utilisons dans ce chapitre

- Ch : représente une chaîne de cluster id. La chaîne (cl_u, Ch)= ($cl_u, cl_0, cl_1, ..., cl_{max}$) avec Ch= ($cl_0, cl_1, ..., cl_{max}$). Cette chaîne contient l'identité du cluster ayant la plus grande identité détectée et la liste de nœuds permettant de l'atteindre. A travers cette chaîne, chaque cluster est à mesure de connaître l'ensemble de ses clusters fils et de son père,
- $longueur(Ch)$: représente la longueur de la chaîne Ch. Elle permet à chaque cluster de choisir comme père le cluster voisin plus près de la racine,
- Soit Ch= ($cl_0, cl_1, ..., cl_{max}$). On note $Ch[0] = cl_0$, $Ch[1] = cl_1$,..., $Ch[longueur(Ch)\text{-}1] = cl_{max}$,

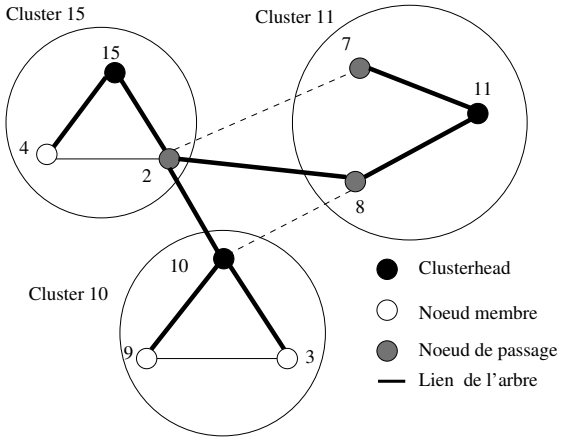


FIGURE 4.8 – Exemple de construction d'arbre

– $Max(Ch)$: représente le cluster id max de la chaîne Ch= $(cl_0, cl_1, ..., cl_{max})$ c'est à dire par construction $Max(Ch) = cl_{max}$.

Un nœud u possède les données suivantes en plus de celles utilisées pour le clustering :
– NPC : représente l'ensemble de couples (cl, id) tels que id est le nœud de passage ayant la plus grande identité permettant de joindre le cluster voisin cl, et id appartient au cluster. Il permet à un clusterhead de choisir un unique nœud de passage pour joindre un cluster voisin,
– C_u : représente un ensemble de chaînes du nœud u. Si $statut_u = CH$, $|C_u| = 1$ et $C_u = \{(Ch_u)\}$. Si $statut_u = NP$, u peut avoir plusieurs chaînes,
– $F(cl_u)$: représente l'ensemble des clusters fils de cl_u,
– $pere(cl_u)$: représente le cluster père de cl_u. Il peut également être déduit de la chaîne Ch.

4.3.3 Sélection des Nœuds de Passage Choisis (NPC)

L'ensemble Nœud de Passage Choisis (NPC_u) est un ensemble de couples (cl, id) tels que id est le nœud de passage ayant la plus grande identité permettant de joindre le cluster $cl \neq cl_u$, et id appartient au cluster de u.
Notre algorithme de construction des NPC se déroule en deux étapes. Dans un premier temps, chaque nœud de passage collecte les identités de ses clusters voisins et transmet à son clusterhead (algorithme 6). Dans un second temps, chaque clusterhead sélectionne parmi les nœuds de passage candidats (s'il y a plusieurs nœuds de passage vers un cluster voisin), le nœud ayant la plus grande identité pour jouer ce rôle (algorithme 7).
– Si u est nœud de passage. u sait s'il existe parmi ses voisins un nœud qui n'appartient pas au même cluster que lui. Donc u collecte les identités de clusters voisins (algorithme 6). Ensuite, il remonte ces informations à son clusterhead.
– Si u est clusterhead. u sélectionne un seul nœud de passage pour chacun de ses clusters voisins. S'il existe plusieurs nœuds de passage candidats vers un cluster voisin, u sélectionne parmi ceux-ci, celui ayant la plus grande identité comme Nœud de Passage Choisis (NPC_u) (algorithme 7). Cette étape nécessite que les informations concernant

Algorithme 6 Construction locale de NPC sur un NP

$NPC \leftarrow NPC_{cl_u} \backslash \{(cl_v, id_v) \in NPC_{cl_u} / id_v = id_u\}$
Pour tout $v \in N_u$ **Faire**
  Si $cl_v \neq cl_u$ **Alors** /* *u est voisin d'un autre cluster* */
    update(NPC_u, $\{(cl_v, id_u)\}$)
  Fin Si
Fin Pour

Algorithme 7 Construction locale de NPC sur un CH

$NPC \leftarrow \emptyset$
Pour tout $v \in N_u$ **Faire**
  Si $cl_v \neq cl_u$ **Alors** /* *u est le nœud de passage vers* cl_v */
    update(NPC_u, $\{(cl_v, id_u)\}$)
  Sinon /* *Récupération des données des NP de mon cluster* */
    update(NPC_u, NPC_v)
  Fin Si
Fin Pour

les clusters voisins soient remontées au clusterhead. Comme l'ensemble NPC est vidé au préalable, le clusterhead ne garde en mémoire que les informations les plus récentes. Les deux étapes de l'algorithme (collection des identités de clusters voisins et sélection des nœuds de passage) ne nécessitent que les informations locales. Le fait que ces étapes soient locales permet une maintenance rapide et permet également à l'algorithme de s'adapter à la mobilité des nœuds.

Algorithme 8 Fonction update(NPC_1, NPC_2)

Pour tout $(cl_u, id_u) \in NPC_2$ **Faire**
  Si $(\nexists (cl_v, id_v) \in NPC_1 / cl_v = cl_u)$ **Alors** /* *Le cluster est inconnu dans mon* NPC_1 */
    $NPC_1 \leftarrow NPC_1 \cup (cl_u, id_u)$
  Sinon /* *Mise-à-jour de* NPC_1 *avec l'identité maximale* */
    Si $id_v < id_u$ **Alors**
      $NPC_1 \leftarrow NPC_1 \backslash \{(cl_v, id_v)\}$
      $NPC_1 \leftarrow NPC_1 \cup \{(cl_u, id_u)\}$
    Fin Si
  Fin Si
Fin Pour

Exemple 4.1 *La figure 4.9 montre la sélection des* NPC*. Dans un premier temps, chaque nœud de passage collecte les identités de ses clusters voisins et remonte ces informations à son clusterhead. Par exemple les nœuds de passage* 5 *et* 6 *envoient à leur clusterhead* 13 *le couples suivants : (11, 6), (16 ; 6) et (16, 5). De la même manière les nœuds de passage* 2 *et* 7 *envoient à leur clusterhead* 16 *le couples (11, 7), (13, 7) et (13, 2). À la réception de ces informations, chaque clusterhead sélectionne ses* NPC *pour joindre ses clusters voisins. Dans notre exemple, le clusterhead* 13 *sélectionne les couples suivants pour joindre ses clusters*

voisins : $NPC_{13} = \{(11, 6), (16, 6)\}$. *De la même manière, le clusterhead* 16 *sélectionne les couples suivants :* $NPC_{16} = \{(11, 7), (13, 7)\}$.

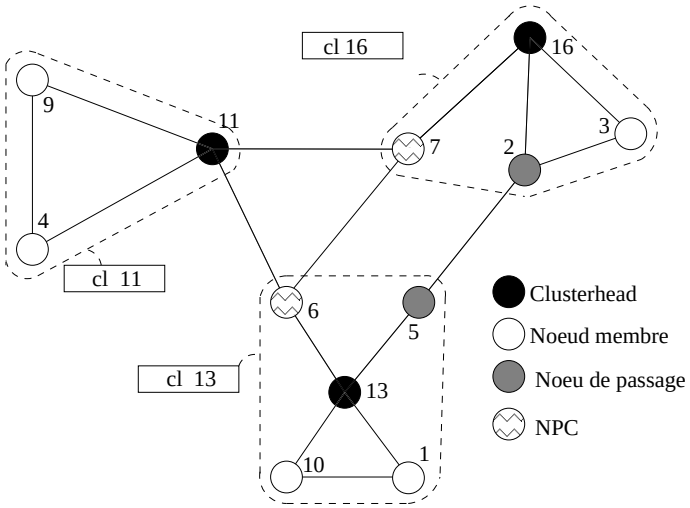


FIGURE 4.9 – Sélection des NPC

4.4 L'algorithme

Dans cette section, nous présentons notre algorithme de construction d'arbre de clusters (algorithme 9 et 10) qui est un algorithme auto-stabilisant. Nous allons tout d'abord commencer par présenter le fonctionnement intuitif avant de le présenter formellement.

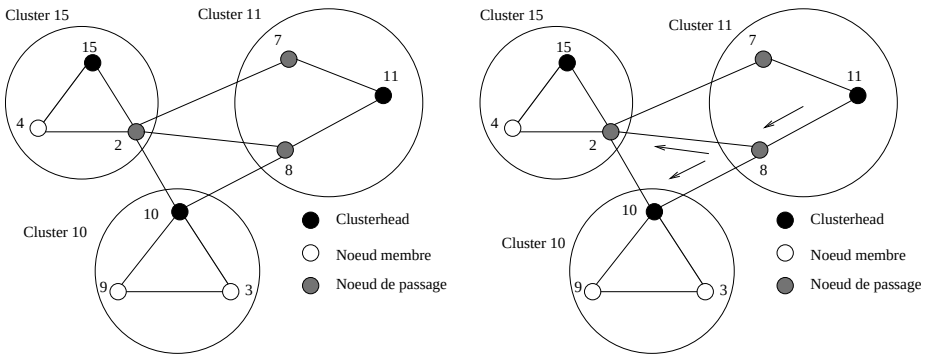
4.4.1 Construction de l'arbre de clusters

La construction de l'arbre de clusters est basée sur les identités associées à chaque cluster. À la fin du processus, le cluster ayant la plus grande identité dans le réseau sera la racine de l'arbre de clusters. La construction de l'arbre de clusters se fait par échange périodique des messages *hello*. Chaque message *hello* transmis par un nœud u contient : id_u, $statut_u$, cl_u, NPC, C_u et $F(cl_u)$. C_u est un ensemble de chaînes contenant les clusters cl d'identités maximales détectées et les listes des clusters permettant de les atteindre et $F(cl_u)$ est l'ensemble de clusters fils de cl_u. Seuls les clusterheads et les nœuds de passage maintiennent les ensembles C_u et $F(cl_u)$. La cardinalité de C_u est égale à 1 pour un clusterhead. Cependant, chaque nœud u exécute l'algorithme de façon distribuée et choisit comme père le cluster ayant la plus grande identité qu'il a détecté (il est aussi la racine pour cl_u). Ensuite, u propage cette information ($C_u = \{(cl_u, pere(cl_u))\}$) à ses clusters voisins. À la réception du message de u par un nœud v, si la plus grande identité détectée par v auparavant est plus petite que celle annoncée par u, v choisit cl_u comme père et propage cette information ($C_v = \{(cl_v, pere(cl_v) = cl_u, pere(cl_u) = racine)\}$) à ses clusters voisins. Ce processus se répète jusqu'à ce que l'arbre soit construit et que le cluster ayant la plus grande identité du réseau soit choisi comme la racine de l'arbre. À la fin du processus, chaque cluster connaît la racine de l'arbre et la liste des clusters permettant de l'atteindre. Chaque cluster connaît également l'ensemble de ses clusters

fils et de son père. Chaque cluster choisit un père parmi les clusters cl de C_u d'un saut au plus proche de lui. Un père est choisi lorsque sa distance au cluster ayant l'identité maximale détectée est plus courte. Par ailleurs, la décision du choix d'un père revient uniquement au clusterhead. Ainsi, un clusterhead n'a pas à attendre la décision des autres clusterheads et peut exécuter l'algorithme de façon distribuée. Notre solution est tolérante aux fautes, prend en compte les changements topologiques en choisissant une nouvelle racine, en mettant à jour l'arbre, en réexécutant l'algorithme sur la topologie modifiée.

Exemple détaillé d'exécution

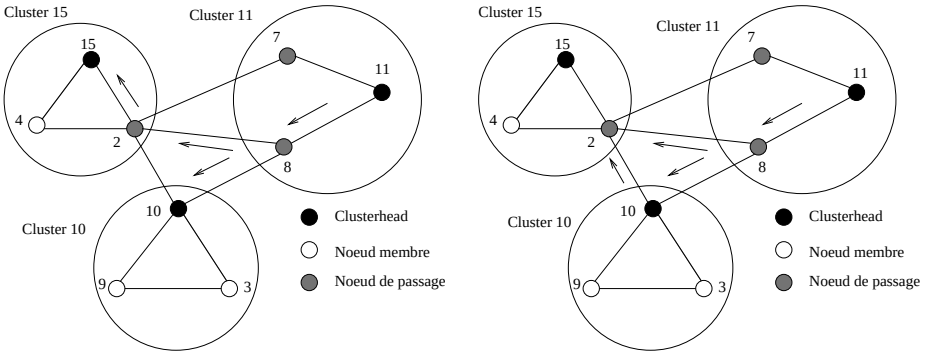
Nous détaillons un exemple d'exécution sur un graphe de clusters. Considérons la topologie de la figure 4.10(a). Cette figure représente la construction de l'arbre suivant notre l'algorithme. Dans un premier temps, chaque clusterhead collecte les identités de ses clusters voisins par l'intermédiaire des nœuds de passage. Ensuite, chacun sélection les Nœuds de Passage Choisis (NPC) en exécutant les algorithmes 6 et 7. Par exemple les nœuds 7 et 8 du cluster 11 permettent d'atteindre le cluster 15. Cependant, le clusterhead 11 sélectionne le nœud 8 comme nœud de passage choisis pour joindre le cluster 15 car il possède la plus grande identité. D'où l'ensemble NPC du clusterhead 11 pour joindre ses clusters voisins sont : $NPC_{11} = \{(15, 8), (10, 8)\}$. Ceux des clusterheads 15 et 10 sont respectivement $NPC_{15} = \{(11, 2), (10, 2)\}$ et $NPC_{10} = \{(15, 10), (11, 10)\}$.



(a) Sélection des NPC clusters (b) Propagation de chaînes

FIGURE 4.10 – Exemple de construction de structure en arbre

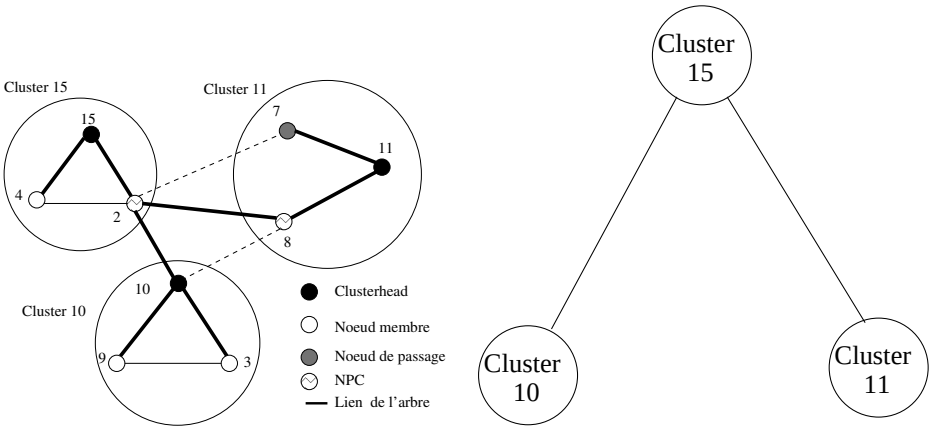
Dans un second temps, les nœuds exécutent l'algorithme de construction de l'arbre (algorithme 9). Considérons la topologie de la figure 4.10(b). La chaîne que le nœud 8 va transmettre aux nœuds 2 et 10 est : $\{(11)\}$ et à la réception de cette chaîne, 2 remonte la chaîne $\{(11)\}$ à son clusterhead (figure4.11(a)).



(a) Propagation de chaînes (b) Propagation de chaînes

FIGURE 4.11 – Exemple de construction d'une structure d'arbre

De la même manière, le clusterhead 11 va recevoir du NPC 8 l'ensemble de chaînes suivantes : $\{(10),(15)\}$. À la réception de ces chaînes, le clusterhead 11 va choisir la chaîne la plus courte vers le cluster ayant l'identité la plus grande détectée (cluster 15). Donc la chaîne $C_{11} = \{(11,15)\}$. De la même manière, le clusterhead 15 va recevoir les chaînes suivantes : $\{(11),(10)\}$. À la réception de ces chaînes, le nœud 15 ignore ces chaînes car le nœud max contenu dans ces chaînes sont tous plus petits que celui de sa chaîne ($C_{15} = \{(15)\}$). Donc le nœud 15 s'élit comme la racine de l'arbre. La chaîne $C_{11} = \{(11,15)\}$ et $C_{10} = \{(10,15)\}$. Cependant le cluster 15 sera le cluster père, 10 et 11 seront ses clusters fils. La figure 4.12 représente l'arbre obtenu.



(a) Construction d'une structure d'arbre (b) l'arbre obtenu

FIGURE 4.12 – La structure d'arbre obtenu

4.4.2 Présentation de l'algorithme

À la réception d'un message *hello*, chaque clusterhead exécute l'algorithme 9. Dans un premier temps, chaque clusterhead vérifie sa cohérence. Un clusterhead u est incohérent si sa chaîne C_u est un ensemble vide ou si son père n'est pas dans son NPC, c'est à dire $pere(cl_u)$ n'est pas un cluster voisin. La règle $R1.a$ permet de corriger cette incohérence en mettant dans

sa chaîne son cluster identité cl_u ($C_u = \{(cl_u)\}$) et en vidant l'ensemble de ses clusters fils ($F(cl_u)$).
Si un nœud u reçoit un message *hello*, il vérifie d'abord s'il est déjà dans la chaîne reçue. Si u n'est pas dans la chaîne reçue alors :

- Si u a reçu la chaîne de la part de son père, il garde la chaîne et ne change pas de père, ni de fils (la règle $R1.b$),
- Si u n'a pas reçu la chaîne de la part de son père mais l'identité du nœud max de la chaîne reçue est plus grande que celle de sa chaîne ou le nœud max de deux chaînes est le même mais la nouvelle chaîne est plus courte, alors u garde la chaîne et choisit comme père le cluster auquel il a reçu la chaîne, et vide l'ensemble de ses clusters fils (la règle $R1.c$),
- Si u n'a pas reçu la chaîne de la part de son père mais l'identité du nœud max de la chaîne reçue est plus petite que celle de sa chaîne ou le nœud max de deux chaînes est le même mais la nouvelle chaîne est plus longue, alors u met à jour ses clusters fils (la règle $R1.d$).

Si u est dans la chaîne reçue et apparaît à la deuxième position de la chaîne, alors u met à jour ses clusters fils (la règle $R1.d$). Sinon il ne traite pas la chaîne.

Algorithme 9 Réception de Ch sur u avec $statut_u = CH$

```
/* Gestion de la cohérence */
Si (C_u = ∅) ∨ (∀w ∈ NPC_u/pere(cl_u) ≠ cl_w) Alors /* La règle R1.a */
  C_u ← {(cl_u)} et F_u ← ∅
Fin Si
/* Traitement de la chaîne Ch */
Si u ∉ Ch Alors
  Si pere(cl_u) = Ch[0] Alors /* La règle R1.b */
    Ch_u ← (cl_u, Ch)
  Sinon Si (Max(Ch) > Max(Ch_u)) ∨ (Max(Ch) = Max(Ch_u)) ∧ (longueur(Ch) + 1) <
  (longueur(Ch_u)) Alors /* La règle R1.c */
    Ch_u ← (cl_u, Ch) et F_u ← ∅
  Sinon /* La règle R1.d */
    F_u ← F_u\{Ch[0]}
  Fin Si
Sinon
  Si id_u = Ch[1] Alors /* La règle R1.e */
    F_u ← F_u ∪ Ch[0]
  Fin Si
Fin Si
```

À la réception d'un message *hello*, chaque nœud de passage u exécute l'algorithme 10.

- Si u a reçu une chaîne de son clusterhead, alors u garde cette chaîne et met à jour son ensemble C_u (la règle $R2.a$),
- Si u a reçu une chaîne d'un nœud appartenant à un cluster voisin et si u a été choisi comme NPC pour ce cluster, alors u garde cette chaîne et met à jour son ensemble C_u (la règle $R2.b$),
- Son ensemble F est égal à celui envoyé par son clusterhead.

Algorithme 10 Réception de Ch_v sur u avec $statut_u = NP$

Si $cl_v = cl_u$ **Alors**
  Si $id_v = cl_u$ **Alors** /* *La règle R2.a* */
    $C_u \leftarrow C_u \cup Ch_v$
  Fin Si
Sinon
  Si $\exists (cl_w, id_w) \in NPC_u / (cl_w = cl_v) \wedge (id_w = id_u))$ **Alors** /* *La règle R2.b* */
    $C_u \leftarrow C_u \cup Ch_v$
  Fin Si
Fin Si

4.4.3 Diffusion d'informations

L'algorithme que nous avons présenté dans ce chapitre permet d'exploiter la structuration locale (le clustering), pour réaliser une structuration globale du réseau (l'arbre couvrant). Cette structure peut aussi être utilisée pour diffuser de l'information dans le réseau. En effet, l'arbre couvrant apporte un avantage dans la diffusion d'informations, car il permet de réduire le nombre de messages échangés et d'atteindre la totalité des nœuds du réseau (au travers les clusters).

Dans cette section, nous présentons un algorithme de diffusion de messages, basé sur l'arbre couvrant de clusters. Comme cet arbre ne contient uniquement que les identités des clusterhead, il est nécessaire de s'intéresser aux rôles des différents nœuds, suivant s'ils sont des clusterheads, des nœuds membres, des nœuds passage. Pour ces derniers, il faut aussi s'intéresser au fait qu'ils soient élus ou non par leur clusterhead comme Nœud de Passage Choisis pour un cluster voisin (voir calcul de l'ensemble NPC). La diffusion peut être divisée en deux étapes : la diffusion globale, qui consiste à envoyer le message le long de l'arbre couvrant de clusters, et la diffusion locale (à un cluster) qui consiste à acheminer le message à un destinataire dans un cluster.

Pour la diffusion locale, les identités des nœuds du cluster ne sont pas connus dans leur globalité suivant le rôle du nœud. Le clusterhead, qui possède un rôle central, permet d'atteindre tous les nœuds de son cluster qui sont maintenus dans son ensemble N (son voisinage). Par contre, un nœud membre ou un nœud de passage ne connaît qu'une partie des nœuds de son cluster, uniquement ceux situés dans leur voisinage. Pour la diffusion locale, les nœuds envoient donc le message au clusterhead qui envoie à son tour le message au destinataire.

Pour la diffusion globale, nous nous basons sur l'arbre couvrant de clusters. Comme nous l'avons montré dans l'algorithme 7, un cluster connaît les identités des clusters à suivre pour atteindre la racine de l'arbre (*i.e.* le nœud d'identité max du réseau). En particulier, l'identité de son père dans l'arbre lui est connu (noté $pere(cl_u) = chu[1]$ si $longueur(ch) > 1$, cl_u si $longueur(ch) = 1$). Tout comme la diffusion classique dans un arbre, un cluster qui désire envoyer un message, l'envoie donc à son père et à ses fils. Lorsqu'un cluster reçoit un message :

- S'il le reçoit de son père, il l'envoie à tous ses fils,
- S'il le reçoit de l'un de ses fils, il l'envoie à son père et à ses autres fils.

Il nous reste maintenant à expliquer comme un cluster (*i.e.* l'un des nœuds du cluster) qui reçoit un message peut l'envoyer à un autre cluster. En effet, un nœud quelconque d'un

cluster qui reçoit un message, doit l'envoyer aux clusters père et fils. Comme nous l'avons dit précédemment, cela passe donc par une étape de diffusion locale puis un acheminement aux clusters voisins. Nous avons ainsi deux solutions :

1. Acheminer automatiquement le message au clusterhead qui l'envoie ensuite aux nœuds de passage nécessaires,
2. Permettre aux nœuds de passage d'acheminer le message à des clusters voisins, si ces clusters sont des fils ou le père de leur cluster,

Pour réduire le nombre de messages et limiter la charge du clusterhead, nous avons choisi la deuxième solution. Notre algorithme général est décomposé en deux algorithmes (algorithmes 7 et 6), suivant si le nœud est clusterhead ou nœud de passage.

Le clusterhead connaît les clusters voisins, ainsi que les nœuds de son cluster qui permettent de les atteindre (éventuellement lui-même) et, de par sa position centrale, il peut atteindre directement tous ces nœuds. Le clusterhead doit vérifier si la destination du message est dans son cluster. Auquel cas il achemine le message directement. Dans le cas contraire, il envoie le message à tous ses clusters voisins dans l'arbre (indifféremment à son père et à ses fils), à l'exclusion du cluster qui lui a envoyé le message. Or, dans le message, nous n'avons pas l'identité du cluster qui a envoyé le message. Par déduction, si c'est un NP d'un cluster voisin, l'identité du cluster de ce nœud est connu dans son ensemble N. Il suffit alors d'envoyer le message à tous les clusters de $F \cup Ch[1]$ privé de l'identité du cluster du NP voisin. Si c'est un NP de son cluster, même avec l'ensemble NPC, il n'est forcément possible de déduire de quel cluster le message provient, comme le montre l'exemple suivant.

Exemple 4.2 *La figure 4.13 montre qu'il n'est pas possible de déduire de quel cluster le message provient, ce qui peut induire une duplication du message. Pour le cluster 8, le nœud de passage choisis vers le cluster 10 est forcément le nœud 4. En effet, c'est l'identité la plus grande parmi les nœuds 2, 4. Pour le cluster 10, c'est le nœud 3 qui est choisi, car il possède l'identité la plus grand parmi les nœuds 1, 3. Si le message provient du nœud 8, il sera envoyé au nœud 4, puis au nœud 1. Si le nœud 1 transfert alors le message au nœud 10, ce dernier peut déduire que le message vient du cluster 9 et non pas du cluster 8.*

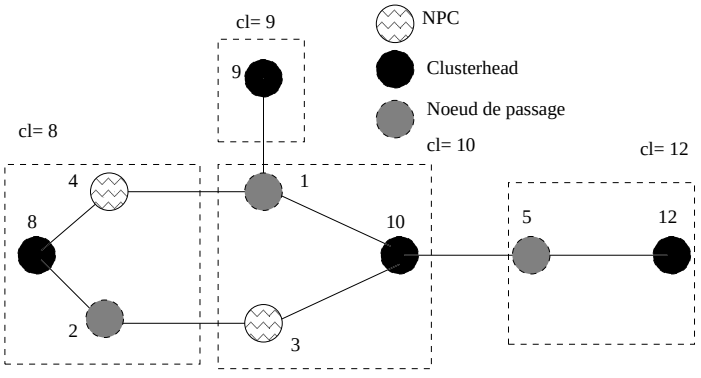


FIGURE 4.13 – Exemple d'une duplication du message

Pour cette raison, nous ajoutons aux messages échangés, l'Identité du Dernier Cluster (IDC) qui a envoyé le message. Cette identité est modifiée lorsque le message doit quitter le cluster et

doit rester identique si le message est diffusé localement.

Lorsqu'un clusterhead reçoit un message d'un cluster voisin, il envoie à tous ses membres de son cluster excepté du nœud par qui il a reçu le message. Si le clusterhead possède des liens vers de clusters voisins, il envoie à tous ses voisins dans l'arbre excepté au dernier cluster visité. S'il possède plusieurs nœuds de passage voisins appartenant au cluster auquel il souhaite envoyer le message, alors il choisit le nœud de passage ayant l'identité la plus grande pour éviter les duplications du message.

Algorithme 11 Réception d'un message (émetteur, données, IDC) sur u avec $statut_u$ = CH du nœud v

Pour tout $w \in N_u$ **Faire**
  Si $id_w \neq id_v$ **Alors** /* *Pas d'envoi vers le nœud qui vient d'envoyer le message* */
    Si $cl_w = cl_u$ **Alors**
      /* *Envoi à tous les nœuds du cluster* */
      Envoyer message (émetteur, données, IDC) à w
    Sinon /* *Le clusterhead possède un lien vers un cluster voisin* */
      Si $cl_w \in (F \cup \{pere(cl_u)\})\backslash\{IDC\}$ **Alors**
        /* *Envoi aux voisins dans l'arbre excepté au dernier cluster visité* */
        Si $\forall x \in N_u\backslash\{w\}, cl_x = cl_w / id_x < id_w$ **Alors**
          /* w *possède la plus grande identité de tous les nœuds de* cl_w*, voisins de* u */
          Envoyer message (émetteur, données, cl_u) à w
        Fin Si
      Fin Si
    Fin Si
  Fin Si
Fin Pour

Lorsqu'un nœud de passage reçoit un message d'un cluster voisin, il peut directement le transférer aux clusters dont il est le NPC (pour rappel, choisi par le clusterhead) et qui sont 1) les fils du cluster ou 2) le père du cluster. Il transfère ensuite le message au clusterhead. S'il possède plusieurs nœuds de passage voisins appartenant au cluster auquel il souhaite envoyer le message, alors il choisit le nœud de passage ayant l'identité la plus grande.

Algorithme 12 Réception d'un message (émetteur, données, IDC) sur u avec $statut_u$ = NP du nœud v

/* *Envoi à tous les clusters voisins de u, excepté le dernier cluster visité, tels que u est nœud de passage vers ces clusters* */
Pour tout $C \in (\{cl_w / w \in NPC_u \wedge id_w = id_u\}\backslash\{IDC\}) \cap (F \cup \{pere(cl_u)\})$ **Faire**
  /* *Envoi au nœud d'identité max parmi les voisins de u qui appartiennent au cluster C* */
  envoyer message (émetteur, données, cl_u) à $Max\{id_w / w \in N_u \wedge cl_w = cl_C\}$
Fin Pour
Si $cl_u \neq id_v$ **Alors** /* *Ce n'est pas le clusterhead qui a envoyé le message* */
  envoyer message (émetteur, données, IDL) à cl_u
Fin Si

4.5 Conclusion

Dans ce chapitre, nous avons proposé un nouvel algorithme de construction d'arbre couvrant pour pouvoir faire de la diffusion dans le réseau. Comme nous l'avons annoncé dans la conclusion 3.9, nous avons exploité notre structure hiérarchique pour proposer une utilisation supplémentaire de la structure. La force de notre solution est qu'il tire parti de la structure hiérarchique. Cependant, l'algorithme de construction de l'arbre ne nécessite que des interactions locales. Dans notre solution, chaque cluster a juste besoin de découvrir les identités de ses clusters voisins et l'identité du cluster max. Chaque nœud n'a pas à attendre la décision des autres nœuds et peut exécuter l'algorithme de façon distribuée. De plus, notre solution est tolérante aux fautes, prend en compte les changements topologiques en choisissant une nouvelle racine, en mettant a jour l'arbre, en réexécutant l'algorithme sur la topologie modifiée. Donc, aucune connaissance globale ni un quelconque GPS n'est nécessaire pour parvenir à faire un arbre couvrant du réseau.

Malheureusement, notre solution risque de surcharger les clusterheads car ils sont responsables de la diffusion d'informations devant transiter par plusieurs clusters. C'est pour cela que nous sommes orientés vers un algorithme de routage adapté à notre solution de clustering qui permet de repartir la charge de diffusion de messages entre les clusterheads et les nœuds de passage.

Une partie des travaux présentés dans ce chapitre a fait l'objet de deux publications en 2010 [FHN10a, FHN10b].

Chapitre 5

Routage

Résumé : *Dans ce chapitre, nous présentons d'abord les différents protocoles de routage dans les réseaux ad hoc. Nous décrivons les trois catégories de protocoles de routage : protocoles proactifs, réactifs et hybrides. Nous présentons ensuite deux solutions de routage tirant partie de la structure en clusters et une solution de routage tirant partie de la structure d'arbre couvrant inter-clusters. Ces deux solutions permettent d'utiliser deux modes de routage différents : un routage proactif au sein des clusters et un routage réactif ou hybride entre les clusters. Pour le routage proactif, chaque nœud maintient une table de routage contenant les membres de son cluster. La partie réactive inter-clusters consiste a envoyé les messages sans initier un processus de découverte de route. Dans cette partie, les nœuds ne conservent aucune donnée en mémoire. La partie hybride inter-clusters consiste à envoyer les données sans initier un processus de découverte de route mais les nœuds conservent des données en mémoire pour des futures transmissions. Ces solutions permettent à la fois réduire le nombre de nœuds chargés de relayer les messages et le délai de bout en bout.*

5.1 Introduction

Lors de la transmission d'un paquet de données d'une source vers une destination, il est nécessaire de faire appel à un protocole de routage qui acheminera correctement le paquet par le meilleur chemin. Le but principal d'un protocole de routage est d'établir une route entre un nœud source et un nœud destination, de sorte que les message soient correctement délivrés dans le réseau. La route doit être établit avec un minimum d'overhead dans le réseau. Comme nous l'avons vu précédemment, un réseau ad hoc est un ensemble de nœuds mobiles qui sont dynamiquement et arbitrairement éparpillés d'une manière où l'interconnexion entre les nœuds peut changer à tout moment. Dans la plupart des cas, l'unité destination ne se trouve pas obligatoirement dans la portée de l'unité source ce qui implique que l'échange des données entre deux nœuds quelconques, doit être effectué par des stations intermédiaires. Pour cela le réseau doit donc s'organiser automatiquement et réagir rapidement aux différents mouvements des nœuds. Chaque unité dévient donc un nœud susceptible d'être mis à contribution pour participer au routage.

Notre point de vue est qu'il faut d'abord organiser le réseau avant de pouvoir router des paquets. C'est dans ce but que nous avons proposé dans le chapitre 3 un algorithme de cluster-

ing permettant d'organiser le réseau en clusters. Cette structure en clusters offre alors une vue simplifiée du réseau et permet de créer une vue logique plus stable. Après avoir présenté dans le chapitre 4 la façon dont notre structure en clusters peut être utilisée pour pouvoir faire de la diffusion dans le réseau. Dans ce chapitre nous présenterons comment une telle structure peut être utilisée pour faire du routage dans le réseau. Notre but est de proposer une solution qui permet à la fois d'optimiser le nombre de messages échangés et le délai de bout en bout. Et en fin permettre le passage à l'échelle.

Nous allons dans un premier temps présenter quelques généralités sur le routage, nous présenterons dans la section 5.4 un état de l'art des solutions de routage dans les réseaux ad hoc. La section 5.9 décrira notre solution de routage.

5.2 Généralités sur le routage

Le routage est une méthode d'acheminement des informations à la bonne destination à travers un réseau de connexion donné. Le problème du routage consiste à déterminer un chemin optimal des paquets à travers le réseau au sens d'un certain critère de performance (bande passante, délai de bout en bout, etc).
Nous allons dans un premier temps faire un rapide rappel sur quelques notions essentielles de routage dans les réseaux avant de nous pencher sur les protocoles de routage dans la section suivante.

Définition 5.1 (Routage par la source et routage par la cible) *Dans les routages par la source, ce sont les nœuds qui émettent les messages qui déterminent la liste des nœuds que les paquets doivent traverser. Par contre, dans le cas du routage par la cible le nœud source signale qu'il souhaite transmettre un message et la destination l'informera de la route à utiliser*

Définition 5.2 (Multihoping) *Dans les communications de type cellulaire les communications passent par des stations dites de base et un réseau filaire : les stations mobiles ne servent jamais de routeurs intermédiaires, ce modèle est donc dit single Hop. Dans un modèle de communication sans ce type d'infrastructures (c'est le cas des réseaux ad hoc), les nœuds participent au routage : le modèle est dit multihop.*

Définition 5.3 (L'inondation) *L'inondation est une technique de routage fréquemment utilisée et qui consiste à faire parvenir un paquet à tous les nœuds du réseau. Ainsi, un nœud qui reçoit un paquet le transmet à tous ses voisins directs. Ce type de routage entraîne une charge importante du réseau et engendre des problèmes tels que les boucles de routage.*

5.3 Taxonomie des protocoles de routage

Les protocoles de routage peuvent être classés suivant plusieurs critères. Ils peuvent être classés selon l'information utilisée pour calculer les routes (vecteur de distance ou état de liens), ou encore selon le type de vision qu'ils ont du réseau et les rôles attribués aux différents nœuds (plat ou hiérarchique). Un troisième critère peut être utilisé pour différencier le protocole de routage est la méthode utilisée pour construire une route entre un nœud source et un nœud

destination (approche réactive ou proactive). Par ailleurs, nous retrouverons cette classification dans les protocoles de routage pour les réseaux ad hoc.

Les protocoles de routage peuvent aussi être classés selon le type de communication, i.e où ils utilisent un
- Unicast,
- Multicast,
- Geocast,
- Broadcast.

La communication unicast est une communication point à point. Il consiste à envoyer un message depuis un hôte vers une destination spécifique. Le broadcast est une communication dans laquelle un message est transmis d'un nœud vers toutes les autres destinations existantes. L'implémentation la plus facile d'un protocole de broadcast est l'inondation aveugle mais cela peut créer une tempête de diffusion due aux retransmissions redondantes. Les protocoles multicast sont utilisés quand un nœud souhaite envoyer un message vers un groupe spécifique de nœuds. Un multicast est nécessaire pour des applications dans lesquelles un sous ensemble des nœuds ont un intérêt commun pour une information spécifique. Dans ce genre de scénario, le multicast est plus performant que unicast en raison de l'économie de bande passante. Les protocoles multicast évitent les transmissions multiples du même message à des récepteurs appartenant au même sous ensemble. Le Geocast est un cas spécial de multicast qui est utilisé pour envoyer un message à un groupe de nœuds situés dans une zone géographique quelconque. Dans le cas de multicast, un nœud peut rejoindre ou quitter un groupe à tout moment, par contre dans le cas de Geocast, un nœud peut rejoindre ou quitter un groupe seulement en entrant ou sortant de la zone géographique correspondante.

5.3.1 États de liens versus Vecteur distance

Les protocoles de routage classiques se classent en deux grandes catégories : protocoles à états de liens et à vecteur de distance.
Dans l'approche à états de liens, chaque nœud maintient une carte plus ou moins complète du réseau où figurent les nœuds et les liens les reliant. Pour construire cette carte, l'algorithme se base sur les informations recueillies sur l'état des liens, ensemble de liens du voisinage dans le réseau. Ces informations sont périodiquement diffusées dans le réseau, ainsi tous les nœuds sont capables au bout d'un certain temps d'avoir une carte topologique complète du réseau. A partir de cette carte topologique, chaque nœud peut calculer de façon indépendante le chemin le plus court vers chaque nœud du réseau. L'algorithme de Dijkstra [Dij59] est utilisé pour calculer le plus court chemin. Par exemple, le protocole Open Shortest Path First (OSPF) [Moy98] est basé sur l'algorithme à états de liens. Bien que cet algorithme permet à chaque nœud d'avoir une vue complète du réseau, cependant, les mises à jour périodiques entrainent rapidement une tempête d'inondation.
Plutôt que de maintenir la carte topologique complète du réseau, l'approche à vecteur de distance ne conserve que la liste des nœuds du réseau et l'identité du voisin par lequel passer pour atteindre la destination par le chemin le plus court. A chaque destination est associé une *direction* et une *distance*. La direction correspond au *next hop* et la *distance* est définie en termes de métrique, comme le nombre de saut. Elle est basée sur un échange périodique

entre nœuds voisins de toutes les destinations connues. Chaque nœud envoie à ses voisins la liste des nœuds qui lui sont accessibles ainsi que le coût correspondant. Les nœuds gardent en mémoire que le coût minimum de chaque destination. Les protocoles à vecteur de distance utilisent généralement l'algorithme Bellman-Ford [BG87] pour déterminer le meilleur chemin. Par exemple, le protocole Routing Information Protocol (RIP) [Mal98] est basé sur l'algorithme à vecteur de distance. Un des inconvénients de cette technique est qu'il présente une convergence lente si la taille du réseau est importante.

Dans les réseaux classiques, les protocoles à états de liens et vecteur de distance donnent souvent des bons résultats à cause des propriétés prévisibles du réseau, comme la qualité des liens et la topologie du réseau. Toutefois, les caractéristiques dynamiques des réseaux ad hoc altèrent leur efficacité. Dans les réseaux ad hoc, quand on utilise un protocole à états de liens ou vecteur distance conçus pour un réseau classique, les changements fréquents de topologie augmentera l'overhead de contrôle. Cela peut influer sur la bande passante du réseau. Ainsi ces protocoles ne sont pas adaptés aux réseaux ad hoc.

5.3.2 Routage à plat versus Hiérarchique

Les protocoles de routage peuvent être classés suivant plusieurs critères. L'un d'entre eux concerne le rôle qu'ils accordent aux différents nœuds du réseau.

Les protocoles de routage à plat

L'approche à plat considère que tous les nœuds du réseau sont dans le même niveau de hiérarchie et possèdent ainsi les mêmes rôles. Par conséquent, aucune hiérarchie n'est définie dans le réseau. La figure 5.3.2 présente un exemple de routage à plat. Dans cet exemple tous les nœuds ont un même rôle qui consiste à relayer les informations vers le nœud suivant.

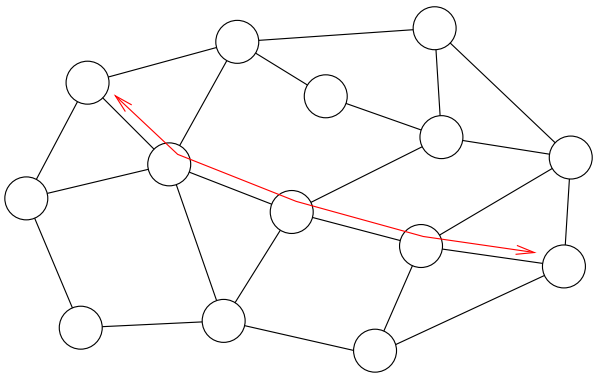


FIGURE 5.1 – Routage à plat

Les protocoles de routage hiérarchique

L'approche hiérarchique attribue aux nœuds des rôles qui varient de l'un de l'autre. Certains nœuds sont élus et assument des rôles particuliers qui donnent une vision hiérarchique au réseau.

Généralement ces approches sont basées sur le découpage du réseau en clusters. Par exemple CBRP [JLT99] propose un routage basé sur une structure en clusters. Des clusterheads sont élus et chargés de coordonner le routage dans leur cluster. Un exemple est donné sur la figure 5.3.2, où une structure en clusters est créée. Cette structure est créée grâce au mécanisme d'élection des clusterheads. Dans cet exemple, si un nœud souhaite envoyer un message, il s'adresse à son clusterhead. Le clusterhead peut donc choisir un nœud de passage pour envoyer le message vers chacun des clusters adjacents. Cette approche permet de ne pas solliciter tous les nœuds du réseau et réduit ainsi la complexité du routage.

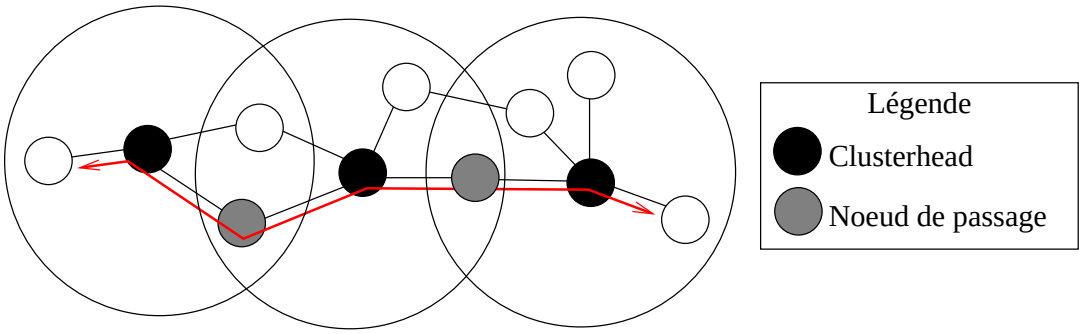


FIGURE 5.2 – Routage hiérarchique

5.4 Protocoles de routage des réseau ad hoc

Toute solution de routage conçue pour un réseau ad hoc doit prendre en considération les propriétés suivantes :

- Minimiser les messages de contrôles (cette propriété impacte la bande passante du réseau),
- Permettre la scalabilité du réseau,
- Minimiser le délai de bout en bout,
- Minimiser les pertes de paquets,
- S'adapter aux changements topologiques (cette propriétés est très importante vue à la dynamicité de la topologie).

Nous décrirons maintenant les différentes familles de protocoles de routage qui existent dans la littérature.

5.4.1 Classification des protocoles de routage

Il existe plusieurs critères pour la conception et la classification des protocoles de routage dans les réseaux ad-hoc : la manière dont les informations de routage sont échangées, quand et comment les routes sont calculées, ... Ainsi, comme indiqué par le schéma en figure 5.3, il est possible de distinguer 3 grandes catégories de routage [AWD03, RT99] :

- Protocoles proactifs : ils établissent les routes à l'avance en se basant sur l'échange périodique de tables de routage,
- Protocoles réactifs : ils recherchent les routes à la demande du réseau. Cette fois lorsqu'un nœud A désire communiquer avec un nœud B, celui-ci commence par demander la con-

struction d'une route vers B en envoyant un message particulier à tous ceux qui peuvent l'entendre,
- Protocoles hybrides : ils combinent les deux approches précédentes afin de tirer avantages de deux catégories précédentes, tout en réduisant leurs inconvénients.

Parmi ces 3 catégories de protocole de routage, certains ont une caractéristique supplémentaire. Il crée ou impose une hiérarchie sur le réseau. Cette approche consiste à "superposer" à la topologie physique, une topologie logique pour le routage.

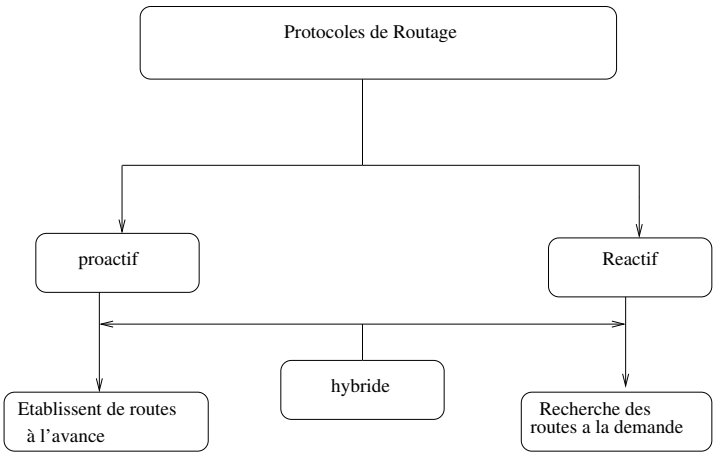


FIGURE 5.3 – Classification des protocoles

5.5 Les protocoles proactifs

Les protocoles de routage proactifs tentent de maintenir à jour dans chaque nœud les informations de routage concernant tous les autres nœuds du réseau [JG07]. Chaque nœud maintien donc une table pour stocker les informations de routage. Du fait de l'aspect dynamique de la topologie des réseaux ad-hoc, la maintenance des tables de routage nécessite l'envoie périodique par chaque nœud un message de signalisation indiquant sa présence à tous ses voisins. L'idée majeure est de conserver dans chaque nœud des informations de routage vers tous les autres nœuds du réseau pour accélérer le routage des paquets par la suite. Les changements topologiques du réseau sont gérés par propagation à chaque voisin des mises à jours des routes afin que chacun puisse maintenir une vue consistante du réseau.

Malheureusement ces protocoles atteignent rapidement leurs limites avec l'accroissement du nombre de nœuds et de leur mobilité. Les changements topologiques sont fréquents. Le réseau sera ainsi constamment inondé par les paquets de contrôle qui réduit considérablement la bande passante. Mais ils permettent, en cas d'envoi successif d'informations d'une même source vers à une même destination, d'utiliser toujours la même route connue à l'avance. Dans le cas d'algorithmes de routages réactifs, le nœud source devrait reconstruire cette route, à plusieurs reprises, pour chaque transmission.

Dans la suite de cette partie, nous présentons les principaux algorithmes de routage proactifs de la littérature.

5.5.1 Destination Sequenced Distance Vector (DSDV)

Le protocole DSDV [PB94] est basé sur l'algorithme distribué de Bellman-Ford. Chaque nœud du réseau maintient dans sa table de routage un ensemble d'information pour chaque destination contenant :

- l'adresse du destination,
- le nombre de saut pour l'atteindre, c'est à dire le nombre de lien de communication existant pour atteindre ces destinations,
- un numéro de séquence associé au destinataire. Il est utilisé pour faire la distinction entre les anciennes routes et les nouvelles routes découvertes vers cette destination pour éviter la formation des boucles de routage.

Afin de maintenir la consistance des tables de routage dans une topologie qui change rapidement, chaque nœud du réseau transmet périodiquement sa table de routage à ses voisins directs. Le nœud peut aussi transmettre sa table de routage si le contenu de cette dernière subit des changements significatifs par rapport au dernier contenu envoyé. Afin de limiter le trafic occasionné par toutes ces mises à jour, il existe deux types de mise à jour :

- des mises à jour complètes,
- des mises à jour incrémentales.

Dans la mise à jour complète, la station transmet la totalité de la table de routage aux voisins. Dans les mises à jour incrémentales, seules les nouvelles entrées ou celles qui ont subit un changement, par rapport à la dernière mise à jour, sont envoyées.

5.5.2 Global State Routing (GSR)

Le protocole GSR [CG98] est similaire au protocole DSDV [PB94] décrit précédemment. Il est également basé sur les états de lien entre nœuds, c'est à dire sur la connaissance que chaque nœud possède de son voisinage. Il utilise ainsi une vue globale de la topologie du réseau. Chaque nœud i maintient une table de voisinage N_i, une table de topologie Top_i, construite par les états de liens de chaque nœud du réseau, une table des nœuds suivants $Next_i$ et une table de distance $Dist_i$. La table de topologie Top_i contient pour chaque destination j l'information de l'état de lien telle qu'elle a été envoyée par j et une estampille pour chaque information. Pour chaque nœud de destination j, la table $NEXT_i$ contient le nœud vers lequel les paquets destinés à j seront envoyés. Finalement, la table de distance D_i contient la plus courte distance pour chaque nœud destination.

Les messages de routage sont générés suivant les changements d'états des liens. Lors de la réception d'un tel message, comme chaque nœud ajoute un identifiant qui lui est propre et qui est incrémenté à chaque transmission, afin de dater les messages, le nœud met à jour sa table de topologie uniquement si le message reçu porte un identifiant plus récent. Par la suite, le nœud reconstruit sa table de routage et diffuse les mises à jour à ses voisins.

5.5.3 Wireless Routing Protocol (WRP)

Le protocole WRP [SGLA96] est basé sur l'utilisation des algorithmes de recherche de chemins appelé Path-Finding Algorithm (PFA). Il en existe de nombreux dans la littérature et

ils utilisent tous le principe suivant : chaque nœud a la connaissance du nombre minimum de lien qui le sépare de tous les autres nœuds. Ces algorithmes impliquent donc de connaître les distances entre chaque nœud du réseau pour pouvoir calculer les plus courts chemin. Dans le protocole WRP, chaque nœud possède cette connaissance, stockée dans une structure appelé table des distances. Chaque nœud possède donc au final :

- une table de distance qui contient la distance connue entre chaque nœud du réseau,
- une table de routage,
- une table de coûts des liens qui contient également les "timeout" associés à chaque nœud,
- et une liste de retransmission de messages qui permet de connaître les nœuds voisins qui n'ont pas acquitté le message de mise à jour et de pouvoir ainsi leur retransmettre.

Les mises à jour sont envoyées à chaque changement d'état d'un des liens voisins ou après réception des données de mise à jour d'un voisin.

Le protocole WRP est caractérisé par la vérification de la consistance des voisins qu'il effectue à chaque fois que le changement d'un lien voisin est détecté. La manière dont il effectue la vérification de cette consistance aide à éliminer les situations des boucles de routage et à minimiser le temps de convergence du protocole.

5.5.4 Clusterhead Gateway Switch Routing (CGSR)

Le protocole CGSR [HKCkW+97] est issu du protocole DSDV [PB94] et est basé sur une architecture de réseau basée sur des groupes. Chaque nœud du réseau est dans un groupe et est d'un des types suivants :

- Clusterhead : c'est le représentant du groupe, qui a pour voisin, tous les autres nœuds du groupe,
- Liaison : ce sont des nœuds communs à plusieurs groupes,
- Sans état : ces nœuds n'ont aucun statut particulier.

Le réseau est ainsi décomposé en groupe, comme illustré sur la figure 5.4.
Afin de s'adapter aux changements du réseau qui sont fréquents dans les réseaux ad-hoc, ce protocole emploie un algorithme appelé Least Cluster Change (LCC). Dans celui-ci, un changement de représentants de groupes n'arrive qu'en cas de fusion de deux groupes, ou bien dans le cas où un nœud sortirait complètement de la portée de tous les représentants du réseau.

Le routage s'effectue de la manière suivante : le nœud source transmet ses paquets de données à son représentant de groupe. Le représentant envoie les paquets aux nœuds de liaison, qui relient ce représentant au représentant suivant dans le chemin qui existe vers la destination. Le processus se répète, jusqu'à ce que le représentant du groupe dans lequel appartient la destination soit atteint. Ce représentant transmet alors les paquets reçus vers le nœud destination.

Chaque nœud maintient deux tables : une table de membre de groupe qui associe à chaque nœud destination son clusterhead et une table de routage qui indique le prochain saut pour atteindre le groupe de destination. Chaque nœud diffuse cette table périodiquement et met la sienne à jour en fonction de celles qu'elle reçoit de la même manière que le fait DSDV. Ce routage est déterministe mais ne donne pas le chemin optimal.

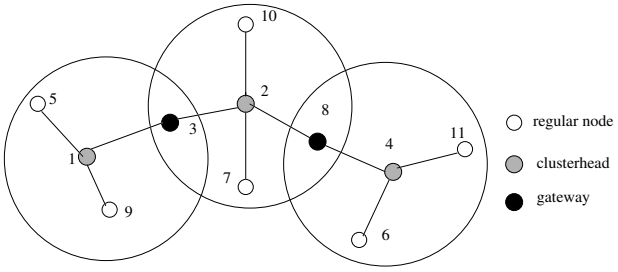


FIGURE 5.4 – Illustration de CGSR

5.5.5 Fisheye State Routing (FSR)

Le protocole FSR [PGC00, GHP02] est une amélioration de GSR [CG98]. Il est basé sur l'utilisation de la technique "oeil de poisson". La grande taille des messages de mise à jour dans GSR gaspille une quantité considérable de la bande passante du réseau. Dans FSR, chaque message de mise à jour ne contient pas l'information sur tous les nœuds. Au lieu de cela, il échange des informations sur les nœuds les plus proches de manière plus fréquente que le font les nœuds les plus lointains. Donc chaque nœud obtient des informations précises concernant les voisins et l'exactitude des informations diminuent avec la distance des nœuds.

Pour le routage, le protocole définit la portée, ou le champ de vision du poisson, en nombre de sauts. Plus un nœud est proche, plus les données maintenues envers celui-ci seront précises. La réduction du volume de données de mise à jour est obtenue en utilisant des périodes d'échanges différentes pour les différentes entrées en fonction de leur distance. Les entrées qui correspondent aux nœuds les plus proches sont envoyées aux voisins avec une fréquence élevée (donc avec une période d'échange relativement petite). Ainsi un grand nombre de données de routage est évité, ce qui réduit le volume des messages qui circule sur le réseau.

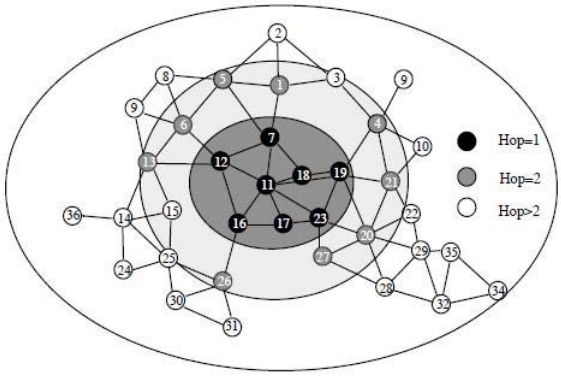


FIGURE 5.5 – La fréquence et la taille des paquets de contrôles sont fonction de la distance dans FSR

5.5.6 Optimised Link State Routing (OLSR)

Le protocole OLSR [JMQ98, CJ03] est un protocole à état de liens qui fabrique des routes de plus court chemin. Contrairement aux algorithmes traditionnels de routage à état de liens dans lesquels chaque nœud diffuse sur tout le réseau les liens directs qu'ils ont avec leurs voisins, dans le protocole OLSR, les nœuds ne déclarent qu'une sous-partie de leur voisinage grâce à la technique des relais multipoints. Ces relais multipoints sont des nœuds qui n'ont la connaissance que de nœuds considérés pertinents. Les nœuds considérés comme redondants pour le calcul des plus courts chemins ne font pas partis de la liste de nœuds connus par ces relais multipoints. Les nœuds pertinents sont sélectionnés de façon à pouvoir atteindre tout le voisinage à deux sauts. Cet ensemble est appelé l'ensemble des relais multipoints.

Le rôle des relais multipoints est de :
- diminuer le trafic engendré par la diffusion des messages de contrôle dans le réseau,
- diminuer le nombre de liens diffusés à tout le réseau puisque les routes sont construites à base des relais multipoints.

Pour maintenir à jour toutes les informations nécessaires au choix des relais multipoints et effectuer le calcul des tables de routage, les nœuds OLSR ont besoin d'échanger des informations périodiquement.

Pour s'informer du proche voisinage, les nœuds OLSR envoient périodiquement des messages de type *hello* contenant la liste de leurs voisins. Ces messages permettent à chacun de choisir son ensemble de relais multipoints. Un deuxième type de message est également utilisé, ce sont des messages appelé *Topology Control*. Par ces messages, les sous-ensembles de voisinage que constituent les relais multipoints sont déclarés périodiquement dans le réseau. Ils sont propagés sur le réseau en utilisant une diffusion optimisée par relais multipoints. Ces informations offrent une carte du réseau contenant tous les nœuds et un ensemble partiel de liens suffisant pour la construction de la table de routage. Cette table est alors calculée par chaque nœud et le routage des données s'effectue saut par saut sans l'intervention d'OLSR dont le rôle s'arrête à la mise à jour des tables de routage de la pile IP.

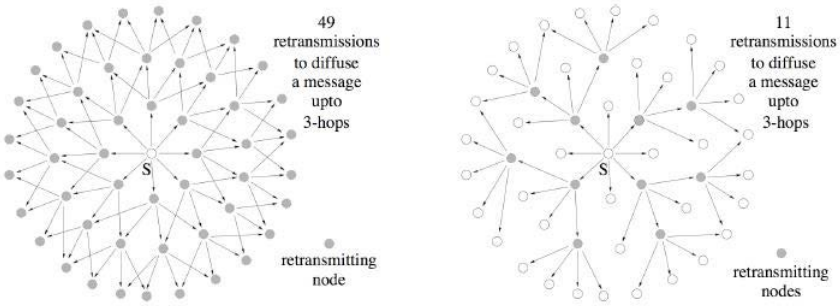


FIGURE 5.6 – Les relais multipoints, base d'OLSR, améliorent l'inondation

5.5.7 Distance Routing Effect Algorithm for Mobility (DREAM)

Le protocole DREAM [BISW98] est basé sur les informations de localisations des unités mobiles. Il diffuse les données destinées à une certaine destination en effectuant une inondation

partielle. Chaque nœud du réseau mobile ad hoc, échange périodiquement des messages de contrôle afin d'informer tous les autres nœuds de sa localisation. La distance influe dans cet échange car les messages de contrôle sont envoyés fréquemment aux nœuds les plus proches. De plus, le protocole s'adapte à la mobilité du réseau par le contrôle de mise à jour de fréquences qui se base sur les vitesses des mouvements.

Lors de l'envoi des données, si la source possède des informations récentes sur la localisation du nœud destination, elle choisit un ensemble de nœuds voisins qui sont localisés dans la direction source/destination. Si un tel ensemble n'existe pas, les données sont inondées dans le réseau entier. Dans le cas où de tels nœuds existeraient, une liste qui contient leurs identifiants est insérée à la tête du paquet de données avant la transmission. Seulement les nœuds qui sont spécifiés dans la liste de tête, traitent le paquet. Lors de la réception du paquet, le nœud de transit, détermine sa propre liste des nœuds prochains et envoie le paquet avec la nouvelle liste de tête. Si aucun voisin n'est localisé dans la direction de la destination, le paquet reçu est ignoré. Quand le nœud destination reçoit les données, il envoie des acquittements à la source d'une manière similaire.

Cependant, dans le cas de réception par inondation, les acquittements ne sont pas envoyés. Dans le cas où la source envoie les données en spécifiant les nœuds suivants (en se basant sur les localisations), un timer associé à la réception des acquittements est activé. Si aucun acquittement n'est reçu avant l'expiration du timeout, les données seront retransmises en utilisant une diffusion.

5.5.8 Topology Broadcast Based on Reverse-Path Forwarding (TBRPF)

Le protocole TBRPF [BR99, OTL04] est un protocole de routage à état de lien conçu pour les réseaux ad-hoc mobiles. Chaque nœud exécutant TBRPF crée un arbre de source fournissant des routes à tous les nœuds accessibles. Il se base sur l'information partielle de topologie stockée dans sa table de topologie, en utilisant une modification de l'algorithme de Dijkstra. Pour réduire au minimum l'occupation de la bande passante, chaque nœud envoi seulement une partie de son arbre de source aux voisins. TBRPF emploie une combinaison de mises à jour périodiques et différentielles pour tenir tous les voisins au courant de la partie rapportée de son arbre source. Chaque nœud a également la possibilité d'envoyer des informations additionnelles de topologie (jusqu'à la topologie complète), pour fournir une fiabilité améliorée dans les réseaux fortement mobiles. TBRPF effectue la découverte du voisinage en utilisant les messages différentiels hello qui rapportent seulement des changements dans le statut des voisins. Ceci a certains avantages car les messages hello sont beaucoup plus petits que ceux utilisés dans d'autres protocoles de routage à état de lien tels que le protocole OSPF.

Nous avons présenté jusqu'ici les protocoles de routages proactifs les plus cités dans la littérature. Mais comme précisé dans l'introduction, il existe une autre grande catégorie de protocoles de routage : les protocoles réactifs, que nous allons maintenant présenter.

5.6 Les protocoles réactifs

Les protocoles réactifs ne gardent que les routes en cours d'utilisation pour le routage. A la demande, le protocole va chercher à travers le réseau une route pour atteindre une destination.

Ces protocoles sont basés sur le principe de la création de route à la demande. Ainsi lorsqu'un nœud souhaite communiquer avec une station distante, il est obligé de déterminer une route dynamiquement. Cette technique permet de ne pas inonder le réseau par des paquets de contrôle et de ne pas conserver les routes non utilisées. Mais elle nécessite en contre partie un certain temps pour établir une route avant de pouvoir la transmettre.

5.6.1 Dynamic Source Routing (DSR)

Le protocole DSR [JHM07, JMJ01] est basé sur l'utilisation de la technique du routage par la source. Dans cette technique la source détermine la séquence complète des nœuds à travers lesquels les paquets de données seront envoyés. Avant d'envoyer un paquet de données vers un autre nœud l'émetteur diffuse un paquet "route request". Si l'opération de découverte de routes est réussie, l'émetteur reçoit un paquet "route response" qui contient une séquence de nœud à travers laquelle la destination peut être atteinte. Le paquet "route request" contient un champ d'enregistrement de routes, dans lequel sera accumulée la séquence de nœud visités durant la propagation de la requête dans le réseau. L'utilisation de la technique de routage par la source fait que les nœud de transit n'ont pas besoin de maintenir les informations de mise à jour pour envoyer les paquets de données, puisque ces derniers contiennent toutes les décisions de routage [Gau03].

DSR est composé de deux mécanismes, le premier est utilisé pour rechercher les routes à la demande et le second s'occupe de la maintenance des routes de communication en cours.

- Le mécanisme de recherche de route est utilisé lorsqu'un nœud source S souhaite envoyer des données à un nœud destination D et que cette route ne figure pas dans le cache de S. Dans ce cas, S inonde le réseau par une requête de recherche de route. Cette requête est ensuite relayée saut par saut vers tous les nœuds en ajoutant à chaque fois dans le message "route request" l'identifiant du nœud courant. Cette inondation se termine quand la destination est atteinte ou bien quand un nœud possède une route vers cette destination dans son cache est trouvé. La liste des nœuds traversés est alors inversée afin de retourner un message de réponse "route response" au nœud S,
- Le mécanisme de maintenance de route permet de signaler au nœud source S que la topologie vient de changer et que la route utilisée pour atteindre D n'est plus valable. Plus clairement, à la suite d'une rupture de lien entre deux nœuds intermédiaires reliant S à D, un message d'erreur est envoyé à S qui va essayer d'utiliser une route alternative de son cache. Sinon il recherche une nouvelle route.

5.6.2 Ad hoc On-demand Distance Vector (AODV)

Le protocole AODV [PE99, PBRD03] représente essentiellement une amélioration de l'algorithme DSR [JHM07, JMJ01]. Il réduit le nombre de diffusions de messages et cela en créant les routes lors du besoin, contrairement à DSR qui maintient la totalité des routes. Il maintient ces routes aussi longtemps que cela est nécessaire au besoin des nœuds sources. AODV utilise un numéro de séquence afin de dater les différentes routes pour avoir des routes à jour. En raison de la mobilité des nœuds dans les réseaux ad-hoc, les routes changent fréquemment ce qui fait que les routes maintenues par certains nœuds deviennent invalides. Les numéros de séquence

permettent d'utiliser les routes les plus récentes. Le protocole AODV est basé sur l'utilisation des deux mécanismes comme DSR :

– Découverte de route,
– Maintenance de route.

AODV [Gau03] construit les routes par l'emploi d'un cycle de requêtes "route request/route reply". Lorsqu'un nœud source désire établir une route vers une destination pour laquelle il ne possède pas encore de route, il diffuse un paquet "route request" (RREQ) à travers le réseau. Les nœuds recevant ce paquet mettent à jour leurs informations relatives à la source et établissent des pointeurs de retour vers la source dans les tables de routage. Outre l'IP de la source, le numéro de séquence courant et un identifiant de diffusion, le paquet RREQ contient également le numéro de séquence de la destination le plus récent connu par la source. Un nœud recevant un paquet RREQ émettra alors un paquet "route reply" (RREP) soit s'il est la destination ou s'il possède une route vers la destination avec un numéro de séquence supérieur ou égal à celui repris dans le paquet RREQ. Si tel est le cas, il envoie un paquet RREP vers la source. Sinon, il re-diffuse le paquet RREQ. Les nœuds conservent chacun une trace des IP sources et des identifiants de diffusion des paquets RREQ. S'ils reçoivent un paquet RREQ qu'ils ont déjà traité, ils le suppriment.

Les nœuds établissent des pointeurs de propagation vers la destination, alors que les paquets RREP reviennent vers la source. Une fois que la source a reçu les paquets RREP, elle peut commencer à émettre des paquets de données vers la destination. Si, ultérieurement, la source reçoit un RREP contenant un numéro de séquence supérieur ou bien le même, mais avec un nombre de sauts plus petits, elle mettra à jour son information de routage vers cette destination et commencera à utiliser la meilleure route.

Une route est maintenue aussi longtemps qu'elle continue à être active. Une route est considérée active tant que des paquets de données transitent périodiquement de la source à la destination selon ce chemin. Lorsque la source arrête d'émettre des paquets de données, le lien expirera et sera alors effacé des tables de routages des nœuds intermédiaires. Si un lien se rompt alors qu'une route est active, le nœud extrémité du lien rompu émet un paquet Route Error (RERR) vers le nœud source pour lui notifier que la destination est désormais non joignable. Après réception du paquet RERR, si la source désire toujours obtenir une route vers cette destination, elle peut re-initier un processus de découverte de route.

5.6.3 Lightweight Mobile Routing (LMR)

Le protocole LMR [CE95] emploie la méthode "source-initialisé", qui construit des routes seulement à la demande de la route. Pour construire une route vers la destination souhaitée, la source inonde un paquet de requête dans le réseau. Notez qu'aucun nœud n'envoie n'importe quel paquet de requête plus d'une fois (les paquets de requête dupliqués peuvent être détectés puisque chaque paquet a une marque unique qui le distingue de tous les autres). La réponse est retournée par un ou plusieurs nœuds qui ont une route vers la destination. On dit qu'un nœud a une route s'il a au moins un lien vers la destination.

Quand la source désire une route vers une destination, elle inonde un paquet de requête dans tout le réseau. En recevant un paquet de requête, si un nœud n'a pas une route, il rediffuse la requête à ses voisins. Si un nœud a une route, il inonde un paquet de réponse vers la source. Notons qu'un paquet de réponse passe seulement par des liens non dirigés, les transformant en

liens dirigés vers l'origine de la réponse. Après que la propagation de réponse est terminée, le protocole LMR construit un ensemble de multiples routes sans boucle enracinée à la destination (DAG) [1].

Le protocole garantit que tous les nœuds participant à l'inondation de réponse obtiennent une ou plusieurs routes. Les routes supplémentaires augmentent la fiabilité. En outre, si l'information de nombre de sauts est ajoutée dans le paquet de réponse, chaque nœud peut améliorer sa décision de routage en choisissant le chemin le plus court. Cependant, dans le protocole LMR, l'optimisation a une importance secondaire ; le principale est de trouver une route rapidement de sorte qu'elle puisse être employée avant que la topologie change. Par conséquent, aucune tentative n'est faite pour maintenir le routage de plus court chemin et ceci élimine le besoin de produire un message de mise à jour quand les évaluations de distance changent.

5.6.4 Temporally Ordered Routing Algorithm (TORA)

Le protocole TORA [VC97] a été conçu principalement pour minimiser l'effet des changements de la topologie. Il s'adapte à la mobilité de ces environnements en stockant plusieurs chemins vers une même destination, ce qui fait que beaucoup de changements de topologie n'auront pas d'effets sur le routage des données, à moins que tous les chemins qui mènent vers la destination soient perdus. La principale caractéristique de TORA est que les messages de contrôle sont limités à un ensemble réduit de nœuds. Dans ce protocole, la sauvegarde des chemins entre une paire (source, destination) donnée ne s'effectue pas d'une manière permanente : les chemins sont crées et stockés lors du besoin, comme c'est le cas dans tous les protocoles de cette catégorie. L'optimisation des routes (i.e. l'utilisation des meilleurs chemins) à une importance secondaire. Les longs chemins peuvent être utilisés afin d'éviter le contrôle induit par le processus de découverte de nouveaux chemins.

L'algorithme TORA appartient à la classe des algorithmes, appelée la classe "Inversement de Liens" (Link Reversal). Il est basé sur le principe des algorithmes qui essaient de maintenir la propriété appelée "orientation destination" des graphes acycliques orientés (DAG). Un graphe acyclique orienté est orienté destination s'il existe toujours un chemin possible vers une destination spécifiée. Le graphe devient non orienté destination si au moins un lien devient défaillant. Dans ce cas, les algorithmes utilisent le concept d'inversion de liens. Ce concept assure la transformation du graphe précédent en un graphe orienté destination durant un temps fini. Afin de maintenir le DAG orienté destination, l'algorithme TORA utilise la notion de taille de nœud. Chaque nœud possède une taille qu'il échange avec l'ensemble de ses voisins directs. Cette nouvelle notion est utilisée dans l'orientation des liens du réseau. Un lien est toujours orienté du nœud qui à la plus grande taille vers le nœud qui la plus petite taille.

Le protocole est basé en trois phases [RT99, VC97] : la création des routes, la maintenance des routes et l'effacement des routes en utilisant trois paquets distincts QRY, UPD et CLR. Chaque nœud i sauvegarde un quintuplet $H_i = (\tau_i, oid_i, r_i, \delta_i, i)$ ou :
τ_i : date d'une rupture
oid_i : identifiant du nœud créateur du niveau de référence
r_i : bit identifiant de réflexion
δ_i : paramètre d'ordonnancement dans la propagation

1. Directed Acyclic Graph

i : identifiant unique du nœud
Les quintuplets sont ordonnés de manière lexicographique.
Chaque nœud i (autre que la destination) maintient sa taille H_i. Au début, chaque nœud i (autre que la destination) place sa taille à NULL, $H_i = (\text{-},\text{-},\text{-},\text{-}, i)$. La taille de la destination est toujours placée à ZERO, $H_{idd} = (0, 0, 0, 0, idd)$, où idd est l'identifiant unique du nœud destination pour laquelle l'algorithme est exécuté. En plus de sa propre taille, chaque nœud i maintient une liste de taille $HN_{i,j}$ pour chaque voisin $j \in i$. Au commencement la taille de chaque voisin est placée à NULL, $HN_{i,j} = (\text{-},\text{-},\text{-},\text{-}, j)$. Si la destination est un voisin de i, le nœud i place l'entrée de la taille de destination à ZERO, $HN_{i,idd} = (0, 0, 0, 0, idd)$.

Le protocole Link-level permet aux nœuds de s'informer des informations relatives à leur voisin. Initié par la source, le processus de découverte de routes pour une destination donnée, crée un DAG orienté vers cette destination. Le nœud source diffuse un paquet QRY spécifiant l'identifiant de la destination, qui identifie le nœud pour lequel l'algorithme est exécuté. Un nœud qui a une taille indéfinie et qui reçoit le paquet QRY, rediffuse le paquet à ses voisins. Un nœud qui a une valeur de taille différente de NULL, répond par l'envoi d'un paquet UPD qui contient sa propre taille. Lors de la réception du paquet UPD, le nœud récepteur affecte la valeur de taille contenant dans le paquet reçu incrémenté de un, à sa propre taille, à condition que cette valeur soit la plus petite par rapport à celles des autres voisins.

5.6.5 Associativity Based Routing (ABR)

Le protocole ABR [Toh97] constitue une nouvelle approche de routage pour les réseaux ad-hoc. Ce protocole introduit la notion de stabilité de route comme nouvelle métrique de routage. Dans ABR, le choix d'une route ne se fait donc plus uniquement sur le nombre de sauts mais tient compte de la stabilité de la connexion existante entre deux nœuds du réseau. L'objectif d'ABR est donc de trouver des chemins ayant une grande durée de vie. Les nœuds génèrent périodiquement des messages de contrôle afin de montrer leur existence aux autres nœuds du réseau. Quand un nœud reçoit un tel signal, il met à jour les valeurs d'associativité avec l'émetteur. Pour chaque signal reçu, un nœud incrémente son intervalle d'associativité qui correspond au nœud émetteur du signal. Les valeurs d'associativité sont remises à zéro quand la connexion entre le nœud est perdue. Une grande valeur d'associativité, correspondant à un nœud voisin et indique un état de faible mobilité de ce nœud. Inversement, une faible valeur indique un état de forte mobilité du voisin.

Ce protocole fonctionne en trois phases principales : la découverte de routes, la reconstruction de routes et la suppression de routes. La phase de découverte de route est un cycle composé d'émission d'une requête (BQ) et d'attente de réponse (BQ-reply). Quand un nœud souhaite trouver un chemin vers une destination, il diffuse un message BQ afin de trouver les nœuds qui mènent vers cette destination. Les nœuds intermédiaires ajoutent leur adresse et leur valeur d'associativité au paquet de la requête et transfèrent le paquet BQ reçu. Ces nœuds de transit ne maintiennent que l'associativité qui leur est associée et celle du nœud qui les précède dans le chemin. De cette manière, chaque paquet qui arrive à la destination va contenir les valeurs d'associativité de tous les nœuds qui appartiennent au chemin reliant la source et la destination. Le nœud destination peut donc choisir le meilleur chemin en comparant les valeurs des différents paquets reçus. Si plusieurs routes ont le même degré global de stabilité d'association, la route avec le nombre minimum de saut est choisie. Une fois ce choix effectué, le nœud des-

tination envoie un paquet de réponse (BQ-reply) au nœud source en utilisant le chemin choisi. Les nœuds qui appartiennent au chemin suivi par le paquet BQ-reply indiquent que leurs routes sont valides, les autres routes restent inactives.

La reconstruction des routes consiste en plusieurs phases : une découverte partielle de routes, une suppression de routes invalides, une mise à jour des routes valides et une nouvelle découverte de routes. Toutes ces étapes ne sont pas effectuées à chaque fois.

Quand un chemin cesse d'être utilisé par une source, une diffusion de suppression de route RD (Route Delete) est lancée. Tous les nœuds appartenant à ce chemin suppriment les entrées correspondantes de leurs tables de routage.

5.6.6 Signal Stability-based Routing (SSR)

Le protocole SSR "Routage basé sur la Stabilité du Signal" [DCWS97], choisit les routes en se basant sur la puissance de signal entre les nœuds, et la stabilité de l'emplacement de ces nœuds. Ce critère de sélection de route a l'effet de choisir les routes qui ont une connexité plus forte.

Le protocole SSR inclut deux protocoles qui coopèrent entre eux : le protocole de Routage Dynamique appelé DRP (Dynamic Routing Protocol) et le protocole de Routage Statique appelé SRP (Static Routing Protocol). Le protocole DRP utilise deux tables : une table de stabilité de signal SST (Signal Stability Table) et une table de routage RT. La table SST sauvegarde les puissances des signaux des nœuds voisins obtenues par l'échange périodique des messages avec la couche de liaison de chaque voisin. La puissance d'un signal est sauvegardée sous l'une de ces deux formes : "canal de forte puissance" ou "canal de faible puissance". Toutes les transmissions sont reçues et traitées par le protocole DRP. Après la mise à jour de l'entrée appropriée de la table, le protocole DRP fait passer le paquet traité au protocole SSR qui consulte sa table de routage pour la destination spécifiée et envoie le paquet reçu au voisin suivant. Si aucune entrée n'existe vers cette destination, le SSR initie un processus de recherche de routes en diffusant un paquet *route request* (RREQ). Le paquet *route request* est envoyée une seule fois (pour éviter le bouclage) et uniquement aux voisins vers lesquels il existe un lien de forte puissance. Le nœud destination choisit le premier paquet *route request* qui arrive car il y a une grande probabilité pour que ce paquet ait traversé le meilleur chemin (le plus court, le moins chargé , etc.) existant entre la source et la destination.

Le protocole DRP du nœud destination inverse le chemin choisi et envoie un message de réponse de route au nœud source. Lors de la réception de cette réponse, le protocole DRP d'un nœud intermédiaire met à jour la table de routage locale suivant le chemin inclus dans le paquet reçu. Les paquets de recherche de routes qui arrivent à destination prennent nécessairement le chemin de forte stabilité de signal car les nœuds de transit n'envoient pas de paquets à travers les liens de faible puissance de signal. Si le délai d'attente de la source (timeout) expire sans avoir reçu de réponse, elle relance de nouveau un processus de recherche de routes en indiquant cette fois ci que les canaux de faibles puissances peuvent être utilisées. Quand une défaillance de liens est détectée sur le réseau, le nœud détectant envoie un message d'erreur au nœud source, en spécifiant le lien défaillant. Lors de la réception de ce message, la source envoie un message de suppression pour avertir tous les nœuds de la défaillance du lien en question. La source initie par la suite un nouveau processus de recherche de routes dans le but de trouver un nouveau chemin vers la destination.

5.6.7 Relative Distance Micro-discovery Ad-hoc Routing (RDMAR)

Le protocole RDMAR [AT99] est basé sur la découverte des distances relatives. Il a été conçu avec pour objectif de minimiser la charge induite par les changements rapides de topologie. Il utilise un mécanisme de découverte de routes. L'idée à l'origine de ce protocole est de diffuser les requêtes qu'à une partie des nœuds : le critère de sélection étant la distance, entre l'émetteur et le destinataire, estimée par un algorithme itératif. Cet algorithme se base sur les informations concernant la mobilité des nœuds, le temps écoulé depuis la dernière communication et l'ancienne valeur de la distance relative. Sur la base de la nouvelle distance calculée, la diffusion de requêtes est limitée à une certaine région du réseau dans laquelle la destination peut être trouvée. Cette limitation de diffusion peut minimiser énormément le contrôle du routage, ce qui améliore les performances de la communication. Dans RDMAR, c'est le nœud destination qui décidera du chemin à prendre. Un nœud qui détecte un problème de lien le signale en diffusant un avertissement.

5.6.8 Core-Extraction Distributed Ad-hoc Routing Algorithm (CEDAR)

Le protocole CEDAR [SSB99] fait partie la famille des protocoles de routage réactifs basé sur une élection dynamique d'un cœur de réseau stable. Le rôle des nœuds du cœur est de propager efficacement des informations sur la bande passante disponible dans les liens, d'assurer le routage dans le réseau en impliquant un minimum de nœuds dans ce processus et de limiter autant que possible les diffusions. CEDAR est basé sur trois composantes essentielles :

- Extraction d'un cœur du réseau : un ensemble de nœud est dynamiquement choisi pour calculer les routes et maintenir l'état des liens du réseau. L'avantage d'une telle approche est qu'avec un ensemble réduit de nœuds les échanges d'information d'état et de route seront minimisés, évitant ainsi plus de messages circulant dans le réseau. En outre, lors d'un changement de route, seuls les nœuds du cœur serviront au calcul,
- Propagation d'état de lien : le routage avec qualité de service est réalisé grâce à la propagation des informations sur les liens stables avec une grande bande passante,
- Calcul de route : celui-ci est basé sur la découverte et l'établissement d'un plus court chemin vers la destination satisfaisant la bande passante demandée. Des routes de "secours" sont utilisées lors de la reconstruction de la route principale, quand cette dernière est perdue. La reconstruction peut être locale (à l'endroit de la cassure), ou à l'initiative de la source.

5.6.9 Location-Aided Routing (LAR)

Le protocole LAR [KN98] est un protocole de routage à la demande s'appuyant sur les informations des localisations fournies par le système de positionnement global GPS pour y découvrir des routes. La nouveauté introduite par les protocoles assistés par un système de localisation (tel que le GPS) est l'utilisation d'une estimation de la position afin d'accroître l'efficacité de la procédure de découverte de route. Le protocole LAR définit deux concepts comme illustre la figure 5.7 :

Expected zone : elle est définie comme la zone où devrait se trouver le nœud du point de vue de la source. L'expected zone à l'instant τ_1 est calculée sur la base de sa position antérieure connue par la source à l'instant τ_0 et de sa vitesse moyenne,

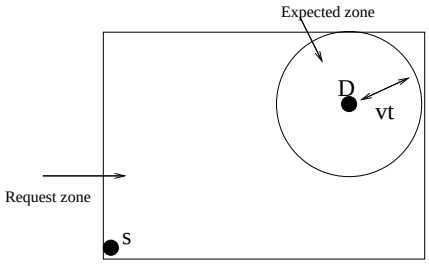


FIGURE 5.7 – Concept de *request zone* et *expected zone* dans le protocole LAR

Request zone : elle est définie comme étant le plus petit rectangle comprenant la position actuelle de la source et de l'expected zone. Cette zone géographique est inscrite dans le message afin d'être exploitée pour limiter l'inondation du réseau lors de l'envoi de la requête de construction de la route : seuls les nœuds contenus dans cette zone prennent en compte le message de recherche de route et l'émettent à leur tour.

5.7 Les protocoles hybrides

Les protocoles hybrides combinent les deux approches. Ils utilisent un protocole proactif, pour apprendre le proche voisinage (voisinage à deux ou trois sauts) et un protocole réactif pour atteindre les nœuds situés au-delà de cette zone prédéfinie de voisinage. Les protocoles hybrides font appels aux techniques des protocoles réactifs pour chercher des routes. Avec ce découpage, le réseau est partagé en plusieurs zones et la recherche de route en mode réactif peut être améliorée. A la réception d'une requête de recherche réactive, un nœud peut indiquer immédiatement si la destination est dans le voisinage ou non et par conséquent savoir s'il faut aiguiller ladite requête vers les autres zones sans déranger le reste de sa zone. Ce type de protocole s'adapte bien aux grands réseaux, cependant, il accumule aussi les inconvénients des protocoles réactifs et proactifs : messages de contrôle périodique, coût de découverte d'une nouvelle route, ...

5.7.1 Zone Routing Protocol (ZRP)

Le protocole ZRP [Zyg97, ZM97] met en place simultanément, un routage proactif et un routage réactif, afin de combiner les avantages des deux approches. Pour ce faire, il passe par un concept de découpage du réseau en différentes zones, appelées "zones de routage". Une zone de routage pour un nœud, est définie par son "rayon de zone". Ce rayon correspond au nombre de sauts maximum qu'il peut y avoir entre deux nœuds.

Le routage au sein d'une zone se fait de manière proactive, via le protocole IARP (Intrazone Routing Protocol) et le routage vers les nœuds extérieurs de la zone se fait de façon réactive, grâce au protocole IERP (Interzone Routing Protocol). En plus de ces deux protocoles, ZRP utilise le protocole BRP (Bordercast Routing Protocol). Ce dernier a pour but de construire la liste des nœuds périphériques d'une zone ainsi que les routes permettant de les atteindre, en utilisant les données de la topologie fournies par le protocole IARP. Il est utilisé pour propager des requêtes de recherche de routes de l'IERP dans le réseau.

La recherche des chemins s'effectue comme suit : on vérifie tout d'abord si le nœud destinataire se trouve dans la zone du nœud source, auquel cas le chemin est déjà connu. Autrement, une demande d'établissement de route RREQ est initiée vers tous les nœuds périphériques. Ces derniers vérifient si la destination existe dans leurs zones. Dans le cas d'une réponse affirmative, la source recevra alors un paquet RREP contenant le chemin menant à la destination. Dans le cas contraire, les nœuds périphériques diffusent la requête à leurs propres nœuds périphériques qui, à leur tour, effectuent le même traitement.

Un exemple de zone est donné à la figure 5.8. Nous remarquons que pour un rayon de zone égal à deux, la zone de routage du nœud S est constituée par tous les nœuds qui sont autour du nœud S avec un maximum de deux sauts les séparant. Sont donc inclus dans la zone de routage, tous les voisins du nœud S ainsi que tous les voisins de ces voisins.

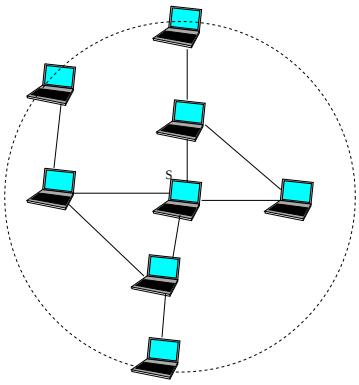


FIGURE 5.8 – Exemple de zone de routage

5.7.2 Zone-Based Hierarchical State (ZHLS)

Le protocole ZHLS [JNL99] est basé sur la décomposition d'un réseau en zone. Contrairement à la plupart des protocoles dit hiérarchiques, il n'y a pas ici de représentant pour chaque zone. La topologie d'un réseau est ainsi partagée en deux niveaux :

- Un niveau nœud indique la façon dont les nœuds d'une zone sont connectés entre eux physiquement. Un lien virtuel peut exister entre deux zones s'il existe au moins un nœud d'une autre zone,
- Un niveau zone qui renseigne sur le schéma de connexion des différentes zones.

Ces niveaux différents entraînent donc deux différents types de liens : les liens inter-nœuds et les liens inter-zones.

Le réseau est donc décomposé comme l'illustre la figure 5.9. Il résulte de cette décomposition un routage inter-zone et un routage intra-zone qui est permise par l'adressage mise en place et qui consiste en un identifiant de zone, un identifiant de nœuds et l'utilisation de LSP (Link State Packet) qui renseignent sur l'état des liens. Il est alors également possible de distinguer deux classes de LSP : ceux orientés nœuds pour lesquels un nœud donné contient des informations sur son voisin et ceux orientés zone qui sont, quant à elles, échangées de manière globale. Ainsi chaque nœud du réseau possède une connaissance complète concernant les nœuds de sa propre zone et seulement une connaissance partielle du reste des nœuds. Les nœuds déterminent

leur position physique en utilisant le GPS. La carte de zone est établie pendant la phase de composition du réseau.

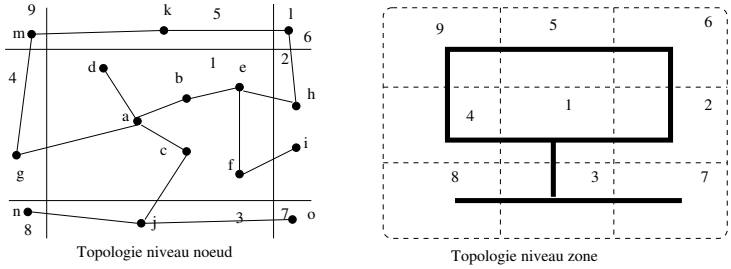


FIGURE 5.9 – Topologie niveau nœud et niveau zone

5.7.3 Hierarchical State Routing (HSR)

Le protocole HSR [PGHC99, ICP+99] définit plusieurs niveaux de hiérarchie dans les nœuds. Le réseau est partitionné en groupes qui se divisent en sous-groupes et ainsi de suite. Un représentant pour chaque groupe est élu. Les représentants des groupes dans un niveau i deviennent des membres dans le niveau $i+1$. Ces nouveaux membres, s'organisent en un ensemble de groupes de la même manière du niveau bas et ainsi de suite pour le reste des niveaux. Les nœuds d'un même groupe se transmettent des informations de routage. Les groupes d'un même niveau communiquent en utilisant un gateway [AWD03, BDDL02].

Chaque nœud a une adresse hiérarchique composé des numéros des groupes sur le chemin de la racine au nœud et cette adresse hiérarchique suffit pour délivrer les paquets de données à une destination, indépendamment de la localisation de la source, simplement en utilisant la table du protocole HSR.

La figure 5.10 illustre le mécanisme de partitionnement dans un réseau de 13 nœuds. Le réseau est décomposé en 3 groupes, qui sont : G0-1, G0-2 et G0-3. Ces groupes forment le niveau le plus bas de la hiérarchie (niveau 0). A partir de ce niveau, les niveaux qui suivent (niveaux 1 et 2), sont formés. Cela est fait en prenant l'ensemble des représentants de groupes et le décomposer en groupes de la même manière que précédemment.

Le nœud représentant d'un groupe donné peut être vu comme un coordinateur de transmission de données. Les identifiants (ID) des nœuds représentés dans la figure 5.10 (niveau 0) sont des adresses physiques : ils sont uniques pour chaque nœud. Une des méthodes qui peut être appliquée afin d'associer des adresses hiérarchiques, ou HIDs (Hierarchical IDs) aux différents nœuds est de prendre les numéros des groupes dans le chemin reliant la racine et le nœud en question. Un nœud de liaison peut être atteint, à partir de la racine, en suivant plusieurs chemins. Par conséquent, ce genre de nœud peut avoir plus d'une adresse hiérarchique.

Prenons l'exemple du routage entre le nœud 6 (source) et le nœud 13 (destination) de la figure 5.10. Les adresses de ces nœuds sont respectivement : $HID(6) =< 1, 1, 6 >$ et $HID(13) =< 2, 3, 13 >$. Pour acheminer une information du nœud 6 vers le nœud 13, le nœud 6 envoie l'information au nœud supérieur, qui le suit hiérarchiquement, i.e. le nœud d'ID 1. Le nœud 1 délivre l'information au nœud 3 qui suit le nœud destination dans l'ordre hiérarchique. Un "lien virtuel" existe entre les nœuds 1 et 3, matérialisé par le chemin $(1, 7, 2, 9, 3)$. Par

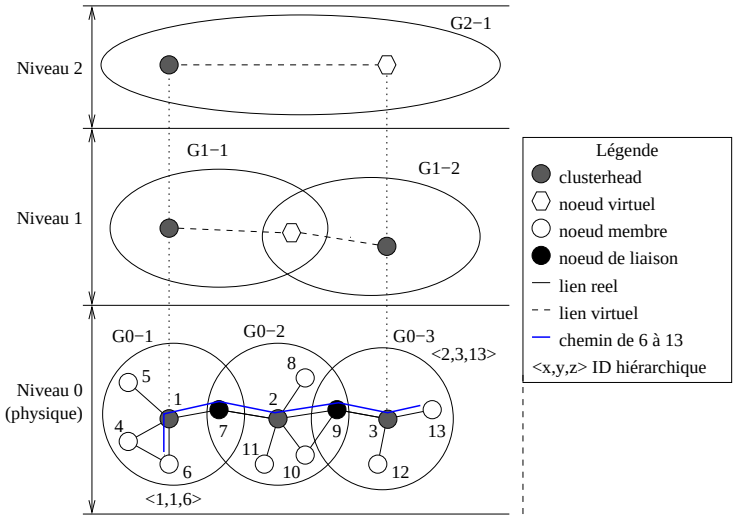


FIGURE 5.10 – Partitionnement du réseau en groupes

conséquent, l'information suivra ce chemin pour atteindre la destination. Dans la dernière étape, le nœud 3 délivre l'information au nœud 13 en suivant le chemin hiérarchique qui le relie à la destination.

5.8 Paramètres pour la comparaison

Cette section introduit les principales notions utilisées dans le tableau 5.1

5.8.1 Type

Le type définit la famille des protocoles. Comme nous avons vu dans les sections précédentes, les protocoles peuvent être classés en 3 grandes familles suivant la manière dont ils créent et maintiennent les routes lors de l'acheminement des données. Les protocoles proactifs, réactifs ou hybride. Les protocoles proactifs établissent les routes à l'avance en se basant sur l'échange périodique de tables de routage alors que les protocoles réactifs recherchent les routes à la demande du réseau. Les protocoles hybrides mélangent les 2 techniques.

5.8.2 Structure

Dans la structure à plat, tous les nœuds du réseau ont le même niveau et ont la même fonctionnalité de routage. Chaque nœud intervient de façon égalitaire dans le réseau, la charge devant donc être répartie de façon uniforme dans le réseau. Apparemment le routage est simple et efficace pour des réseaux de petites tailles. Le problème survient dans des réseaux de grande dimension, le volume d'informations de routage devenant important, il devient important de localiser des nœuds cibles.

Dans la structure hiérarchique, les nœuds du réseau sont regroupés en cluster donnant ainsi une vue différente de la topologie du réseau dans sa forme normale. Dans cette structure, des leaders de zones sont créés et ils sont chargés de coordonner le routage dans le réseau. Elle

permet d'optimiser la diffusion d'information dans le réseau en limitant le nombre de nœuds chargés de stocker l'information et de relayer des paquets. Cette structure est bien adaptée aux réseaux de grandes tailles.

5.8.3 Routes

Certains protocoles de routage trouvent un itinéraire unique entre un nœud source et un nœud destinataire. D'autres protocoles découvrent plusieurs routes pour une même destination. Ils ont ainsi l'avantage de pouvoir proposer une alternative en cas de modification topologique, sans faire de redécouverte de routes. En outre, la source choisit la meilleur route au cas où plusieurs routes existent entre un nœud source et un nœud destination.

5.8.4 Routage par la source

Le routage par la source est la détermination complète de la route par la source. Dans cette technique, le chemin total (c'est à dire les nœuds de passage) est contenu dans l'en-tête de paquet de données afin que les nœuds intermédiaires seulement relayent les paquets en fonction de chemin spécifié dans l'en-tête des paquets. L'avantage de cette technique est que les nœuds intermédiaires n'ont pas besoin de maintenir les informations de mise à jour afin d'acheminer les paquets de données qu'ils transmettent puisque les paquets eux-mêmes contiennent toutes les décisions de routage. Le plus gros problème de routage par la source est lorsque le réseau est vaste et que la route est longue, la mise de chemin dans l'en-tête de paquet va consommer beaucoup de bande passante [ZRM02].

Le routage nœud après nœud consiste à ne donner que l'adresse du prochain nœud pour accéder au destinataire. Quand un nœud reçoit un paquet à une destination, il transmet le paquet au nœud suivant correspondant à la destination. Le problème est que tous les nœuds nécessitent de maintenir les informations de routage

5.8.5 Messages de contrôle

De manière générale, les messages de contrôle sont de petits messages nécessaires au transport des informations de topologie. Ce sont ces messages qui sont à la base des information permettant le routage. Ils sont diffuser généralement par inondation. De manière plus spécifique, les messages hello sont des messages envoyés par tous les nœuds afin d'annoncer leur présence aux nœuds voisins et ainsi construire les tables de voisinage à 1 saut et dans quelques cas à deux sauts [2] . Les protocoles basés sur des informations topologiques utilisent souvent des messages de contrôle diffusés sur tout ou partie du réseau, certains d'entre eux (OLSR par exemple) utilisent les deux.

5.8.6 Information géographique

Certains protocoles de routage utilisent des GPS ou d'autres systèmes de coordonnées pour pouvoir diriger leur message vers une destination. Ces protocoles basent le calcul de routes

2. C'est -à-dire que deux communication sont nécessaires pour atteindre le destinataire

sur la position, supposée connue des nœuds. Ces routages sont en général utilisés dans des environnements dynamiques et où le besoin de passage à l'échelle est nécessaire.

Protocoles	Type	Structure	multi-routes	Routage par la source	Message de contrôle	Information géographique
DSDV	proactif	plat	simples	non	non	non
CGSR	proactif	hiérarchique	simples	non	non	non
FSR	proactif	plat	simples	non	non	non
OLSR	proactif	plat	simples	non	oui	non
DREAM	proactif	plat	simples	oui/non	oui	oui
TBRPF	proactif	plat	simples	non	oui	non
WRP	proactif	plat	simples	non	oui	non
GSR	proactif	plat	simples	non	non	non
LAR	réactif	plat	simples	non	non	oui
SSR	réactif	plat	simples	non	oui	non
ABR	réactif	plat	simples	oui	oui	non
LMR	réactif	plat	multiples	non	oui	non
RDMAR	réactif	plat	simples	non	non	non
AODV	réactif	plat	simples	non	oui	non
DSR	réactif	plat	multiples	oui	non	non
CEDAR	réactif	hiérarchique	simples	oui	oui	non
TORA	réactif	plat	multiples	non	oui	non
ZRP	hybride	plat	multiples	oui pour interzone	oui	non
ZHLS	hybride	hiérarchique	simples	non	non	non
HSR	proactif	hiérarchique	simples	non	non	non

TABLE 5.1 – Comparaison des protocoles proactifs / réactifs

[H]

Protocole	Noeud critique	Methode	Metriq. routage	temps de converg.	complex.en temps	compléx. de com.
DSDV	non	broadcast	PCH	O(D.I)	O(D)	O(N)
CGSR	oui, CH	broadcast	PCH	O(D)	O(D)	O(N)
FSR	non	broadcast	PCH	O(D.I)	O(D)	O(N)
OLSR	non	broadcast	PCH	O(D.I)	–	–
DREAM	non	multicast	RD	O(N.I)	–	–
TBRPF	oui, NP	broadcast	PCH	O(D)/(D+2)	–	–
WRP	non	broadcast	PCH	O(h)	O(D)	O(N)
GSR	non	broadcast	PCH	O(D.I)	O(D)	O(N)
LAR	non	broadcast	RD	PCH	O(S)	O(2M)
SSR	non	unicast	ASL & SS	–	O(D+Z)	O(N+R)
ABR	non	broad./unicast	ASL & PCH	–	O(D+Z)	O(N+R)
LMR	non	broadcast	PCH	–	O(2D)	O(2A)
RDMAR	non	–	PDR & PCH	–	O(2S)	–
AODV	non	unicast	PF & PCH	–	O(2D)	O(2N)
DSR	non	unicast	PCH	–	O(2D)	O(2N)
CEDAR	non	broadcast	PCH	–	O(D)	O(C+D)
TORA	non	broadcast	PCH	–	O(2D)	O(2N)
ZRP	non	broadcast	PCH	–	O(M)/O(2D)	O(M)/O(2B*D)
ZHLS	non	broadcast	PCH	–	O(M)/O(D)	O(M)/O(N)
HSR	oui,CH	broadcast	PCH	O(D)	O(D)	O(N)

CH= clusterhead ; NP= nœud parent ; PCH= plus court chemin ; RD= routage directionel ; PF= plus frais ; PDR= plus courte distance relative ; AS= associativité de lien ; SS= stabilité du signal ; D= diamètre du réseau ; I= intervalle moyen de mise à jour ; h= hauteur de l'arbre de routage ; N= nombre de nœud dans le cluster ; S= diamètre de région où localisée les nœuds ; M= nombre moyen des nœuds dans la zone ou cluster ; A= nombre de nœuds affectés ; R= Nombre de nœuds formant le chemin RRP ; B= nombre moyen de nœuds gateway d'une zone ou clusterhead

TABLE 5.2 – Comparaison des protocoles proactifs / réactifs

5.9 Notre solution

Comme nous l'avons vu précédemment, la majorité des protocoles se base sur une topologie à plat où tous les nœuds ont des rôles égalitaires et utilisent massivement les inondations : les protocoles réactifs pour envoyer des requêtes de demande de routes et les protocoles proactifs pour envoyer des requêtes de topologie. Il est en effet très probable que les performances d'un tel réseau seront réduites vue que l'inondation entraine une charge importante du réseau et engendre des problèmes tels que : le gaspillage de bande passante à cause des duplications de messages, sur consommation énergétique, ...

Pour ces raisons, nous proposons deux solutions de routage différentes :

1. Une solution de routage basée sur une structure en clusters. Cette solution est décomposée en deux parties :
   - Réactive inter-cluster et proactive intra-cluster,
   - Hybride inter-cluster et proactive intra-cluster.
2. Une solution de routage basée sur une structure d'arbre couvrant.

La partie réactive inter-cluster consiste à envoyer les données sans initier un processus de découverte de route. Dans cette partie, les nœuds ne gardent aucune donnée en mémoire pour des futures transmissions sauf la communication du cluster. Chaque nœud maintient une table contenant les membres de son cluster et utilise cette table pour faire du routage proactif à l'intérieur de son cluster.

La partie hybride consiste à envoyer les données sans initier un processus de découverte de route mais cette fois-ci les nœuds conservent des données en mémoire pour des futures transmissions. Donc, les nœuds construisent leur table de routage au fur et à mesure qu'ils transmettent des données. De la même manière, les communications intra-clusters utilisent un protocole de routage proactif. Notre objectif est à la fois d'optimiser le délai de bout en bout et le nombre de messages échangés. Pour ces raisons, nous combinons dans une même phase la construction des routes et la transmission de données.

La deuxième solution consiste à utiliser un arbre couvrant pour pouvoir faire du routage. Le but est de réduire la taille de la table de routage au niveau de chaque nœud. Cependant, cette solution nécessite une étape de construction d'arbre avant de pouvoir router les données.

Nous allons dans un premier temps décrire le fonctionnement général des solutions proposées. Puis nous décrirons en détail le routage réactif inter-cluster et routage proactif intra-cluster, le routage hybride inter-cluster et proactif intra-cluster. Ensuite, Nous présenterons le routage basé sur une structure d'arbre couvrant. Enfin, nous présenterons notre algorithme de routage

5.9.1 Description générale

Nous proposons de router les informations en utilisant deux solutions différentes :
- Un protocole de routage basé sur une structure en clusters,
- Un protocole de routage basé sur une structure d'arbre couvrant.

La première solution est décomposée en deux parties :

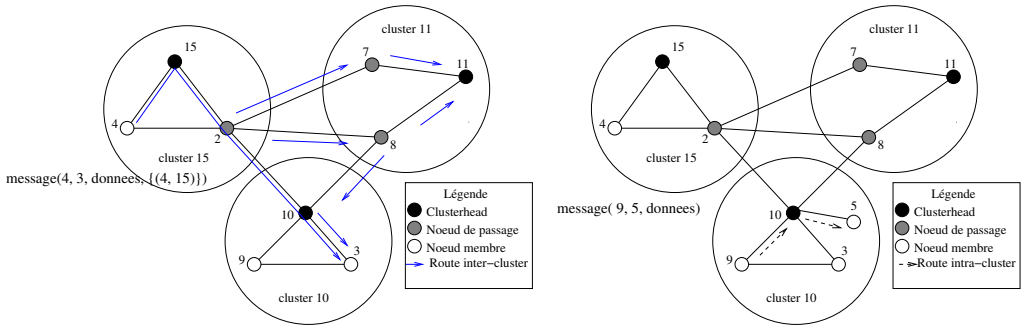
1. Réactive inter-cluster et proactive intra-cluster,
2. Hybride inter-cluster et proactive intra-cluster.

La partie réactive inter-cluster consiste à envoyer les données sans initier un processus de découverte de route. Cette technique permet de ne pas inonder le réseau par des paquets de découverte de route et la maintenance de celle-ci sans parler des pannes qui risquent de compromettre les chemins construits. Elle permet également d'optimiser le délai de bout en bout en combinant dans une même phase la découverte et la transmission de données. Quand un nœud souhaite envoyer un message et que la destination n'est pas dans son cluster alors il l'envoie à ses clusters voisins en ajoutant son identité et son cluster *id* dans une liste LP, cette liste est insérée dans l'entête du message. Elle contient le parcours complet du message. Elle permet également d'éviter les boucles de routage. Ce processus se répète jusqu'à ce que le message arrive au destinataire. Cependant, les nœuds ne conservent aucune donnée en mémoire pour des futures transmissions. Chaque nœud possède une connaissance proactive au sein de son cluster et utilise un protocole proactif pour les communications intra-cluster. Quand un nœud souhaite envoyer un message à un nœud de son cluster, si la destination est un voisin alors il le transfert directement au destinataire. Dans le cas contraire, il l'envoie à son clusterhead et celui-ci le transfert directement au destinataire.

Remarque 5.1 *La liste LP permet d'éviter les boucles de routage. Dans le cas de routage réactif inter-cluster, seuls les clusters id peuvent être utilisés. Cependant, pour optimiser l'entête du message, on peut seulement ajouter le cluster id dans la liste LP.*

Exemple 5.1 *Dans la figure 5.11(a), le nœud* 4 *souhaite envoyer un message au nœud* 3*. Le nœud source et le nœud destination ne sont pas dans le même cluster. Le protocole réactif inter-cluster sera utilisé pour envoyer le message au nœud* 3*. Donc le nœud* 4 *l'envoie à son clusterhead en ajoutant dans la liste LP son identité et son cluster id. Ensuite, le clusterhead ajoute à son tour son identité et son cluster id dans la liste LP et l'envoie à ses nœuds de passage. Chaque nœud de passage transmet ensuite le message aux autres clusterheads voisins et aux autres nœuds de passage voisins. Le message n'est relayé que par les nœuds de passage et les clusterheads.*

Dans la figure 5.11(b), le nœud 9 *souhaite envoyer un message au nœud* 5*. Le nœud source et le nœud destination sont dans le même cluster. Cependant, le nœud* 9 *et le nœud* 5 *ne sont pas voisins. Donc le nœud* 9 *l'envoie à son clusterhead* 10 *et celui-ci le transfert directement au nœud destinataire.*



(a) Routage inter-cluster (b) Routage intra-cluster

FIGURE 5.11 – routage inter-cluster réactif et intra-cluster proactif

La partie hybride inter-cluster consiste à envoyer les données sans initier un processus de découverte de route mais chaque nœud qui relaie un message conserve une trace des nœuds constituant le parcours du message. Parallèlement à la transmission des données, les nœuds construisent leur table de routage contenant les distances et les passerelles pour atteindre les destinations déjà rencontrées. Plus un nœud relaie des messages plus sa table sera complète car celle-ci est mise à jour à chaque réception de message. Quand un nœud souhaite envoyer un message et qu'il ne possède aucune entrée dans sa table de routage vers cette destination, il l'envoie à ses clusters voisins en ajoutant son identité et son cluster *id* dans la liste LP. La liste LP permet aux nœuds de construire leur table de routage. Elle permet également d'éviter les boucles de routage. À la réception du message, chaque nœud met à jour sa table et relaie le message en ajoutant son identité et son cluster *id*. Ce processus se répète jusqu'à ce que le message arrive au destinataire. Nous conservons la trace des routes utilisées sur chaque nœud à destination des nœuds extérieurs aux clusters. Les nœuds connaissent le chemin vers les destinations déjà rencontrées. Ainsi les nœuds connaissent les éléments suivants :

– La passerelle (le nœud par lequel il faut passer pour atteindre la destination),
– La distance jusqu'au ce nœud.

Chaque nœud possède une connaissance proactive au sein de son cluster et utilise un protocole proactif pour les communications intra-cluster. Quand un nœud souhaite envoyer un message à un nœud de son cluster, si la destination est un voisin alors il le transfert directement au destinataire. Dans le cas ou un nœud possède une entrée dans sa table vers ce nœud, il l'envoie à la passerelle correspondante et celle-ci le transfert directement au destinataire.

Remarque 5.2 *Dans le cas du routage hybride inter-cluster, la liste LP contient tous les nœuds qui relaient le message ainsi que leur cluster id. Ces informations sont nécessaires pour permettre aux nœuds de construire leur table de routage.*

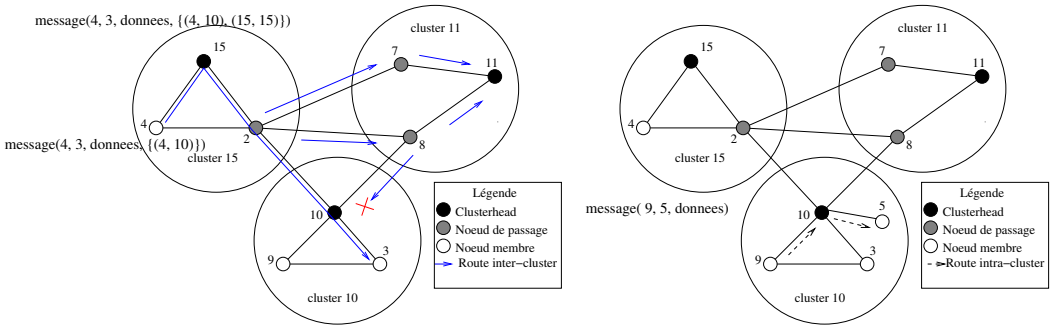
Exemple 5.2 *Dans la figure 5.12(a), le nœud* 3 *souhaite envoyer un message au nœud* 3*. Le nœud* 3 *et* 4 *ne sont pas dans le même cluster et le nœud* 3 *ne possède aucune entrée vers le nœud* 4*. Il l'envoie à son clusterhead en ajoutant dans l'entête du message son identité et son cluster identité. le clusterhead envoie à ses nœuds de passage en ajoutant dans l'entête du message son identité et son cluster identité. La route entre* 4 *et* 3 *passe par les clusterheads* 15 *et* 10*. Le clusterhead* 15 *envoie le message (4, 3, données, {(4, 15), (15, 15)}) au nœud* 2*. À la réception, le nœud* 2 *ajoute à son tour son identité, son cluster id et l'envoie aux clusters*

voisins. Donc 2 *envoie le message (4, 3, données,* {*(4, 15), (15, 15), (2, 15)*}*) aux nœuds* 7, 8 *et* 10. *Ce processus se répète jusqu'à ce que le message arrive à destination. Donc le nœud* 3 *recevra le message suivant : (4, 3, données, message*{*(4, 15), (15, 15), (2, 15), (10, 10)*}*). Le message n'est relayé que par les nœuds de passage et les clusterheads.*

Par exemple dans la figure 5.12(a), quand le nœud 10 *relaie le message vers le nœud* 3, *le nœud* 10 *sait que la distance pour atteindre le nœud* 4 *est de* 3 *sauts, et la passerelle vers celui-ci est le nœud* 2. *Il sait aussi que la distance pour atteindre le nœud* 15 *est de* 2 *sauts, et la passerelle vers celui-ci est le nœud* 2. *Ainsi quand le nœud* 3 *reçoit le message, il sait que la distance pour atteindre le nœud* 4 *est de* 4 *sauts, et la passerelle vers celui-ci est le nœud* 10. *Il sait aussi que les distances pour atteindre les nœuds* 2 *et* 15 *sont respectivement de* 2 *et* 3 *sauts. Par exemple le nœud* 3 *maintient la table de routage suivante :*

id	distance	suivant
2	2	10
15	3	10
4	4	10

TABLE 5.3 – Table de routage du nœud 3



(a) Routage inter-cluster (b) Routage intra-cluster

FIGURE 5.12 – routage inter-cluster hybride et intra-cluster proactif

Dans la figure 5.12(b), le nœud 9 *souhaite envoyer un message au nœud* 5. *Le nœud source et le nœud destination sont dans le même cluster. Cependant, le nœud* 9 *et le nœud* 5 *ne sont pas voisins. Donc le nœud* 9 *l'envoie à son clusterhead* 10 *et celui-ci le transfert directement au nœud destinataire.*

La deuxième solution consiste à utiliser un arbre pour pouvoir faire du routage. Cette technique permet de supprimer les duplications de messages et de réduire la taille des tables de routage. De la même manière que le protocole hybride, parallèlement à la transmission des données, les nœuds construisent leur table de routage contenant les distances et les passerelles pour atteindre les destinations déjà rencontrées.

Chaque nœud possède une connaissance proactive au sein de son cluster et utilise un protocole proactif pour les communications intra-cluster. Quand un nœud souhaite envoyer un message à un nœud de son cluster, si la destination est un voisin alors il le transfert directement au destinataire. Dans le cas contraire, il l'envoie à son clusterhead et celui-ci le transfert directement au destinataire.

Exemple 5.3 *Dans la figure 5.13, le nœud* 4 *souhaite envoyer un message au nœud* 9. *Les nœuds* 4 *et* 9 *ne sont pas voisins et le nœud* 4 *ne possède aucune entrée vers le nœud* 9. *Le nœud* 4 *ajoute dans l'entête du message son identité et son cluster identité, envoie le message (15, 9, données,* {*(4, 15)*}*) à son clusterhead* 15. *Ensuite, le clusterhead* 15 *l'envoie à ses clusters fils. Donc le clusterhead* 15 *envoie le message (15, 9, données,* {*(4, 15), (15, 15)*}*) au nœud* 2. *À la réception, le nœud* 2 *envoie le message (15, 9, données,* {*(4, 15), (15, 15), (2, 15)*}*) aux nœuds* 8 *et* 10. *Quand le nœud* 9 *reçoit le message, il sait que la distance pour atteindre le* 4 *est de* 4 *sauts et la passerelle pour atteindre celui-ci est le nœud* 10. *Le nœud* 11 *sait aussi que la distance pour atteindre le nœud* 4 *est de* 4 *sauts et la passerelle pour atteindre celui-ci est le nœud* 8.

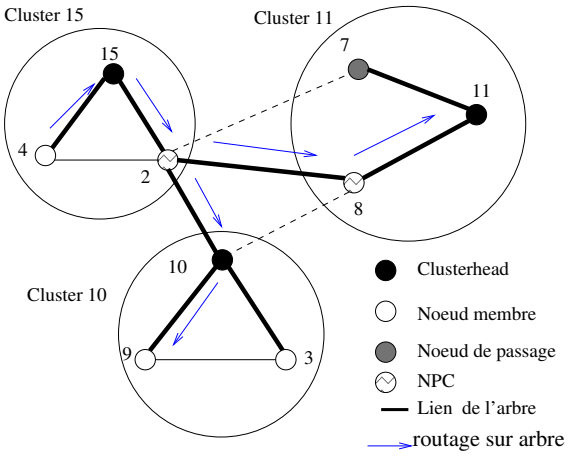


FIGURE 5.13 – routage sur un arbre couvrant

5.9.2 L'algorithme

Dans cette section, nous présentons notre solution de routage basée sur la structure en clusters (algorithmes 13 et 14). Nous allons d'abord commencer par présenter le fonctionnement intuitif avant de le présenter formellement.

Nous introduisons ici les données sur les nœuds et les données des messages.

1. Données sur les nœuds.

– *LC* : table des membres du cluster. Elle permet à un nœud de maintenir une connaissance proactive au sein de son cluster.

1. Données des messages.

- LP : liste des nœuds où le message est passé ((id_A, cl_A), (id_B, cl_B), ...). Cette liste permet aux nœuds de construire leur table de routage. Elle permet également d'éviter les boucles de routage,
- *destination* : destination du message,
- *source* : source du message,
- structure de la table de routage : $TR(id_e, distance_e, suivant_e)$, id_e est l'identité du nœud e, $distance_e$ correspond aux nombres de sauts nécessaires pour atteindre e, $suivant_e$ est le nœud de passage pour atteindre e depuis le nœud u,
- structure d'un message à envoyer : $envoi(source, destination, données, LP)$.

Routage proactive intra-cluster

Nous utilisons un protocole proactif à l'intérieur des clusters. Grâce à notre structure hiérarchique, nous pouvons introduire, sans créer de surcoût important un protocole proactif au sein des clusters. Afin de maintenir une connaissance locale dans les clusters, chaque nœud du réseau maintient une table LC qui contient les membres du cluster. Chaque nœud construit sa table en se basant sur les informations reçues à partir de son clusterhead et par échange périodique de messages *hello* utilisé dans l'algorithme de clustering. En plus de ces messages *hello*, le clusterhead envoie périodiquement sa table aux nœuds membres de son cluster. Contrairement à l'algorithme CGSR 5.5.4, où chaque nœud diffuse périodiquement sa table des membres du cluster, dans notre solution, les nœuds membres et les nœuds de passage ne diffusent pas leur table. Seul le clusterhead envoie par unicast ou multicast sa table à ses membres. En plus de cette table, chaque nœud maintient un ensemble N (voisinage).

Exemple 5.4 *Par exemple dans la figure 5.14, le nœud* 7 *maintient la table des membres du cluster ci-dessous. Ainsi, avec ces deux tables, chaque nœud est en mesure de savoir si la destination est dans son voisinage ou dans son cluster. Si par exemple un nœud u souhaite envoyer un message à un nœud v qui est dans le même cluster que lui mais pas dans son voisinage, alors u l'envoie à son clusterhead.*

Dans la figure 5.14, le nœud 7 *souhaite envoyer un message au nœud 8. Les deux nœuds ne sont pas voisins mais ils sont dans le même cluster. Le nœud* 7 *l'envoie à son clusterhead et celui-ci le transfert directement au destinataire.*

id	8	11
cl	11	11
Statut	Nœud de passage	Clusterhead

TABLE 5.4 – Table des membres du nœud 9

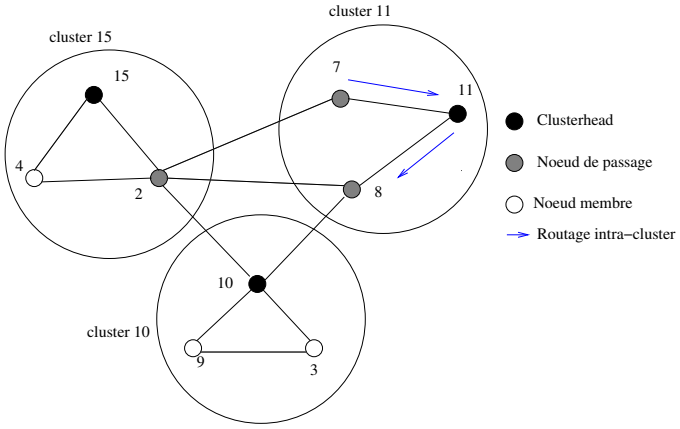


FIGURE 5.14 – Routage intra-cluster

Routage inter-cluster sans mémoire

Nous utilisons un protocole réactif pour les communications inter-cluster. Contrairement à la plupart des algorithmes existants qui utilisent deux phases pour le routage inter-cluster : la construction des routes et la transmission de données, notre solution combine dans une même phase la découverte de routes et la transmission de données. Elle consiste à envoyer les données sans initier un processus de découverte de route. De cette façon, le nœud source n'a pas besoin d'initier la découverte de route et la maintenance de celle-ci. Elle permet également de ne pas inonder le réseau par des paquets de découverte de route et la maintenance de celle-ci. Dans cette technique les nœuds ne conservent aucune donnée en mémoire pour des communications futures. Quand un nœud souhaite envoyer un message à un nœud destination qui n'est pas dans son cluster, il l'envoie à ses clusters voisins. Ce processus se répète jusqu'à ce que le message arrive au destinataire. Pour éviter les boucles, nous ajoutons la liste LP dans l'entête du message. Grâce à notre structure hiérarchique, la diffusion est optimisée en limitant le nombre de nœuds chargés de relayer les paquets. Par contre, il peut y avoir une duplication de messages.

Supposons qu'un nœud u souhaite envoyer un message à un nœud v. Si v n'est pas dans le cluster de u, u doit envoyer le message contenant l'adresse de destination aux clusters voisins. Ainsi nous avons trois cas :

- Si u est un clusterhead, alors il l'envoie à tous les nœuds de passage voisins,
- Si u est un nœud membre, alors il l'envoie à son clusterhead et celui-ci l'envoie à tous ses nœuds de passage voisins,
- Si u est un nœud de passage, alors il l'envoie à son clusterhead et aux clusters voisins.

Exemple 5.5 *Dans la figure 5.16, le nœud* 9 *souhaite envoyer un message au nœud* 4 *mais le nœud destination* 4 *n'est pas dans le même cluster que lui. Donc le nœud* 9 *envoie le message contenant l'adresse de destination à son clusterhead* 10 *et celui-ci l'envoie aux nœuds de passage voisins, nœuds* 2 *et* 8*. Ainsi le nœud* 2 *l'envoie directement au nœud destinataire* 4*. La figure montre que le nœud* 2 *transfert trois fois le même message. Cela est dû du fait que les nœuds ne gardent aucune donnée en mémoire.*

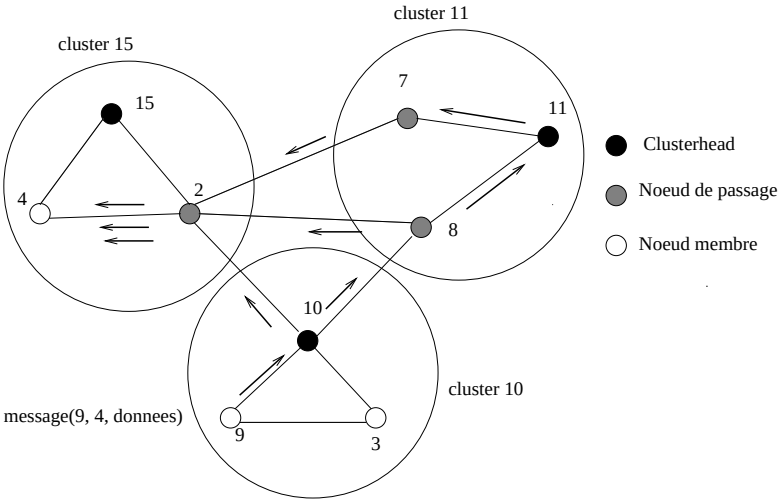


FIGURE 5.15 – Routage inter-cluster

Routage hybride inter-cluster avec mémoire

Dans cette partie nous ne décrivons pas la partie proactive intra-cluster. Cette partie est la même que celle décrite dans la section 5.9.2. Dans [Lav90, Lav86], l'auteur a proposé un algorithme basé sur le mot circulant.
La partie hybride inter-cluster consiste à envoyer les données sans initier un processus de découverte de route mais les nœuds construisent leur table de routage parallèlement à la transmission de messages. Notre solution combine dans une même phase la découverte des routes, la transmission de données et la construction des tables de routage. Les nœuds construisent leurs tables de routage au fur et à mesure qu'ils transmettent les messages. Sur réception d'un message, chaque nœud intermédiaire met à jour sa table de routage. Chaque entrée de cette table est associée à un nœud extérieur au cluster déjà rencontré, elle contient l'identité du nœud, la distance pour atteindre ce nœud et la passerelle vers ce nœud. Ainsi, chaque nœud qui relaie un message conserve une trace des nœuds constituant le parcours du message. Ces informations sont obtenues grâce à la liste LP contenue dans l'entête du message. Chaque nœud qui relaie un message ajoute à la liste LP son identité et son cluster identité. Cette liste contient le parcours complet du message. Il faut aussi préciser que dans le routage inter-cluster, on utilise deux modes de routage différents : routage proactif et réactif. La partie réactive concerne le routage inter-cluster. Grâce à notre structure hiérarchique, la diffusion est optimisée en limitant le nombre de nœuds chargés de relayer les paquets. On utilise un protocole proactif si un nœud souhaite envoyer un message et s'il possède une entrée dans sa table de routage vers cette destination. Comme mentionné précédemment, plus un nœud reçoit de messages, plus sa table sera complète.
Quand un nœud souhaite envoyer un message à un nœud destination et si celui-ci ne possède aucune entrée vers cette destination, il l'envoie en ajoutant son identité et son cluster id à ses clusters voisins. Ce processus se répète jusqu'à ce que le message arrive au destinataire. Chaque nœud qui relaie le message met à jour sa table de routage.

Supposons qu'un nœud u souhaite envoyer un message à un nœud v. Si u ne possède aucune entrée vers le nœud v, u doit l'envoyer en ajoutant son identité et son cluster id aux clusters

voisins. Ainsi nous avons trois cas :
- Si u est un clusterhead, alors il l'envoie à tous les nœuds de passage voisins,
- Si u est un nœud membre, alors il l'envoie à son clusterhead et celui-ci envoie à tous ses nœuds de passage voisins,
- Si u est un nœud de passage, alors il l'envoie à son clusterhead et à tous les nœuds voisins appartenant aux clusters voisins.

A chaque fois qu'un nœud reçoit un message, il vérifie s'il possède une entrée dans sa table de routage vers cette destination. Si c'est le cas, le protocole proactif est exécuté pour envoyer directement le message à la passerelle correspondante. Sinon le protocole réactif est exécuté afin d'envoyer le message au destinataire. Ce protocole réduit le trafic réseau et optimise les routes en éliminant la phase de découverte de routes. Dans ce mode de routage, nous ajoutons une optimisation qui permet de réduire les duplications de messages. Après transmission d'un message :
- Si un message qui a emprunté un chemin plus cours arrive par la suite, il le transfert aussi,
- Dans le cas contraire, il l'ignore.

Exemple 5.6 *Dans la figure 5.16, le nœud* 9 *souhaite envoyer un message au nœud* 4 *mais il ne possède aucune entrée dans sa table de routage vers cette destination. Donc le nœud* 9 *envoie le message en ajoutant on identité et son cluster id à son clusterhead* 10*. Donc le nœud* 9 *envoie le message (9, 4, données,* $\{$*(9, 10)*$\}$*) au nœud* 10*. Le nœud* 10 *envoie à son tour (9, 4, données,* $\{$*(9, 10), (10, 10)*$\}$*) aux nœuds de passage voisins, nœuds* 2 *et* 8*. Ainsi le nœud* 2 *le transfert directement au nœud destinataire* 4 *et ajoute une entrée dans sa table de routage vers le nœud* 9*. A la réception du message, le nœud* 4 *ajoute également une entrée dans sa table de routage vers les nœuds* 9 *et* 10*. C'est ainsi que les nœuds construisent leur table de routage. Dans cet exemple, on considère que le nœud* 2 *a reçu le message du nœud* 10 *avant ceux des nœuds* 7 *et* 8*, donc il ignore les messages provenant des nœuds* 7 *et* 8 *car ces messages ont emprunté des chemins plus longs. Même si le nœud* 2 *ne transfert pas ces messages, il met à jour sa table de routage. Par contre si un message qui a emprunté un chemin plus long est arrivé en premier, il peut y avoir une duplication du message. Après transmission d'un message, si un message qui a emprunté un chemin plus cours arrive par la suite, il le transfert aussi. Par exemple dans la figure 5.16, si on suppose que le nœud* 2 *a reçu le message provenant du nœud* 8 *avant celui du nœud* 10*, alors le nœud* 2 *transfert deux fois le même message mais dans ce cas, cela permet d'optimiser le contenu des tables de routage. Donc il peut y arriver qu'un nœud transfert deux fois le même message. Après transmission du message, tous les nœuds qui ont reçu le message complètent leur table de routage. Par exemple dans la figure 5.16, le nœud* 2 *maintient la table de routage suivante (table 5.6). A la réception du message par le nœud* 4*, la liste LP contenu dans l'entête du message est :* $\{(9, 10), (10, 10), (2, 15)\}$.
Par exemple le nœud 2 *maintient la table de routage suivante :*

id	distance	suivant
9	2	10
11	2	8
11	2	7

TABLE 5.5 – Table de routage du nœud 2

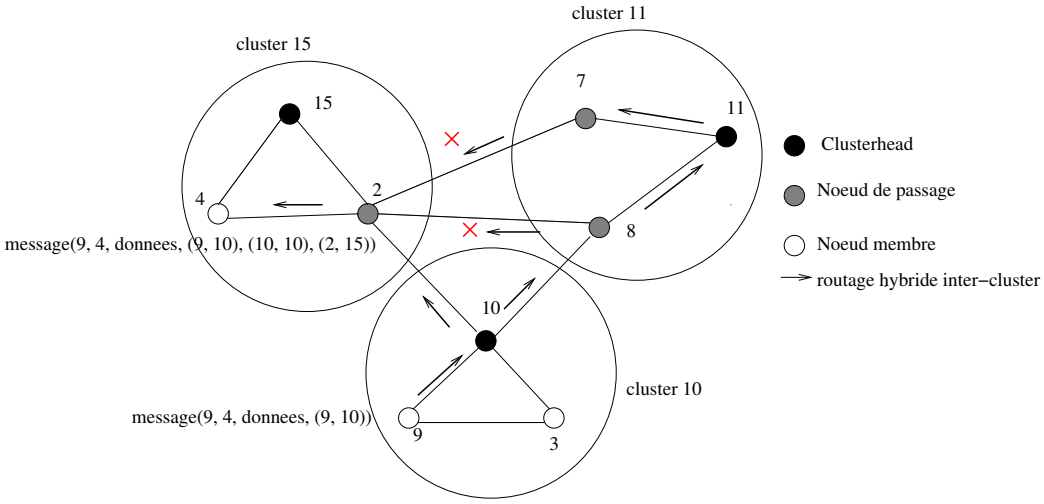


FIGURE 5.16 – Routage hybride inter-cluster

Routage hybride sur arbre

Dans cette partie nous ne décrivons pas la partie proactive intra-cluster. Cette partie est la même que celle décrite dans la section 5.9.2.
De la même manière que le routage hybride inter-cluster avec mémoire, les nœuds envoie les données sans initier un processus de découverte et ils construisent leur table de routage parallèlement à la transmission des données. Sur réception du message, chaque nœud met à jour sa table de routage.
Quand un nœud souhaite envoyer un message à un nœud destination et que s'il ne possède aucune entrée vers cette destination, il l'envoie à son cluster père et à ses clusters fils en ajoutant son identité et son cluster id dans la liste LP. Chaque cluster connaît l'identité de son père dans l'arbre et celles de ses fils. Lorsqu'un cluster reçoit un message :

- S'il reçoit de son père, il l'envoie à tous ses fils,
- S'il reçoit de l'un de ses fils, il l'envoie à son père et tous ses autres fils.

Supposons qu'un nœud u souhaite envoyer un message à un nœud v. Si u ne possède aucune entrée vers le nœud v, u doit l'envoyer à son cluster père et/ou à ses clusters fils en ajoutant son identité et son cluster id dans la liste LP. Ainsi nous avons quatre cas :

- Si u est un clusterhead et s'il possède des voisins dans l'arbre, il l'envoie en ajoutant son identité et son cluster id à tous ses voisins dans l'arbre,
- Si u est un nœud membre, alors il l'envoie à son clusterhead et celui-ci l'envoie à tous ses voisins dans l'arbre,

– Si u est un nœud de passage, alors il l'envoie à son clusterhead et celui-ci l'envoie à tous ses voisins dans l'arbre,
– Si u est un Nœud de Passage Choisi (NPC), alors il l'envoie à son clusterhead et à tous ses voisins dans l'arbre.

Exemple 5.7 *Dans la figure 5.17, le nœud* 9 *souhaite envoyer un message au nœud* 4 *mais il ne possède aucune entrée dans sa table de routage vers le nœud* 4. *Donc le nœud* 9 *envoie le message (9, 4, données, {(9, 10)}) à son clusterhead* 10. *Le cluster* 10 *envoie le message (9, 4, données, {(9, 10), (10, 10)}) à son cluster père* 15. *Ainsi, le nœud* 2 *transfert directement au destinataire* 4 *et ajoute une entrée dans sa table de routage vers le nœud* 9. *À la réception, le nœud* 4 *ajoute également une entrée vers le nœud* 10 *et le nœud* 9.
Par exemple le nœud 4 *maintient la table de routage suivante :*

id	distance	suivant
10	2	2
9	3	2

TABLE 5.6 – Table de routage du nœud 2

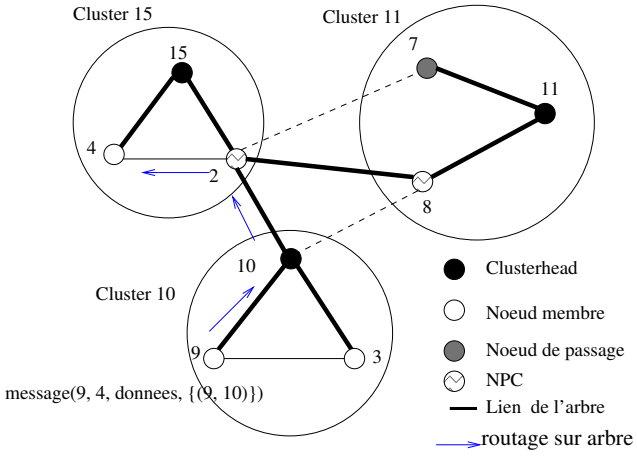
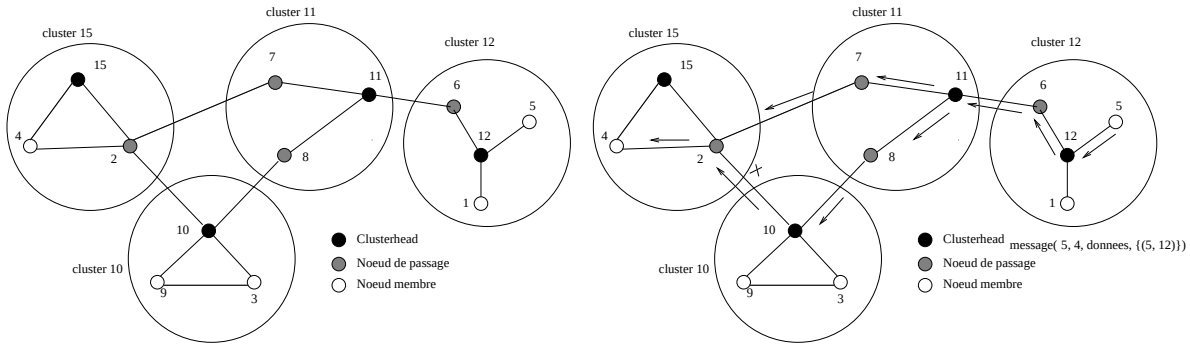


FIGURE 5.17 – routage sur un arbre couvrant

Exemple détaillé d'exécution

Nous détaillons ici un exemple d'exécution de notre algorithme avec mémoire sur un graphe de 12 nœuds. Cet exemple décrit le routage inter-cluster avec mémoire selon le statut du nœud source (clusterhead, nœud membre ou nœud de passage). Dans la figure 5.18, le nœud source (avec $statut(Source) = NM$) souhaite envoyer un message au nœud destinataire. De plus, le nœud ne connaît aucune route vers la destination. Alors le nœud source l'envoie à son clusterhead et le clusterhead l'envoie aux clusters voisins. A la réception du message, si le destinataire n'est pas dans la table de routage alors le message sera envoyé aux clusters voisins. Ainsi, ce processus se répète jusqu'à ce que le message arrive au destinataire. Dans cet exemple, on suppose que le nœud 2 a reçu le message du nœud 7 avant celui du nœud 10. À

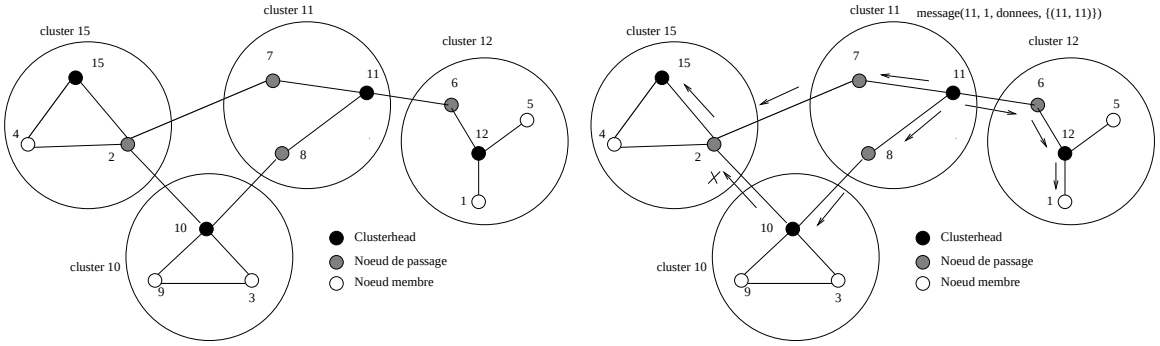
la réception du message par le nœud 4, la liste LP contenu dans l'entête du message est : $\{(5, 12), (12, 12), (6, 12), (11, 11), (7, 11), (2, 15)\}$.



(a) Topologie (b) Routage de données

FIGURE 5.18 – Le nœud source est un nœud membre

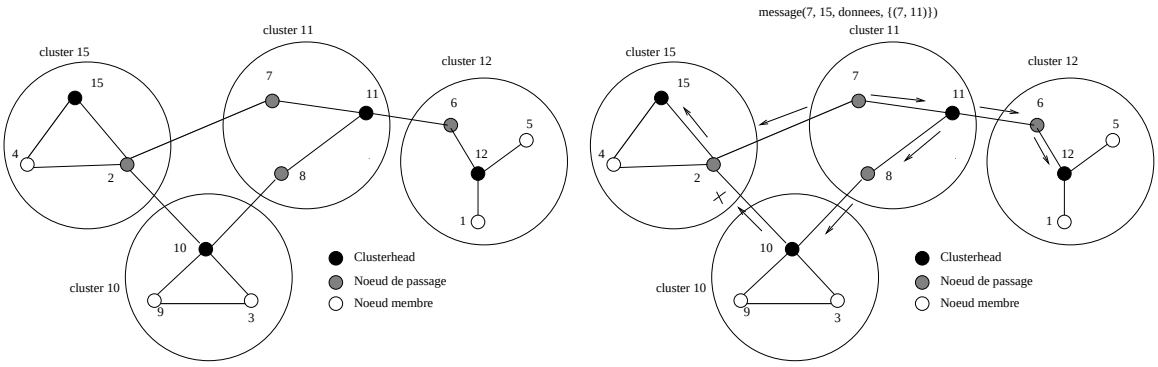
Dans le figure 5.19, le nœud source (avec $statut(source) = CH$) souhaite envoyer un message au nœud destinataire. De plus, la source ne connait aucune route vers la destination. Alors le nœud source l'envoie aux clusters voisins. À la réception du message, si le destinataire n'est pas dans la table de routage alors le message sera envoyé aux clusters voisins. Ainsi, ce processus se répète jusqu'à ce que le message arrive au destinataire. À la réception du message par le nœud 1, la liste LP contenu dans l'entête du message est : $\{(11, 11), (6, 12), (12, 1)\}$.



(a) Topologie (b) Routage de données

FIGURE 5.19 – Le nœud source est un clusterhead

Dans la figure 5.20, le nœud source (avec $statut(source) = NP$) souhaite envoyer un message au nœud destinataire. De plus, la source ne connait aucune route vers la destination. Alors le nœud source l'envoie aux clusters voisins et à son clusterhead. A la réception du message, si le destinataire n'est pas dans la table de routage alors le message sera envoyé aux clusters voisins. Ainsi, ce processus se répète jusqu'à ce que le message arrive au destinataire. À la réception du message par le nœud 15, la liste LP contenu dans l'entête du message est : $\{(7, 11), (2, 15)\}$.



(a) Topologie (b) Routage de données

FIGURE 5.20 – Le nœud source est un nœud de passage

5.9.3 Présentation de l'algorithme

Mise à jour de la table de routage. Chaque nœud u qui souhaite envoyer un message ajoute dans l'entête du message les information suivantes : son identité et son cluster identité dans la liste LP. Chaque nœud qui relaie ce message ajoute à son tour ces informations et conserve une trace des nœuds constituant le parcours du message. Ces informations permettent à chaque nœud intermédiaire de construire sa table de routage. Chaque entrée de cette table est associée à un nœud extérieur déjà rencontré, elle contient l'identité du nœud, la distance pour atteindre ce nœud et la passerelle vers ce nœud.

Envoi d'un message. Quand un nœud u souhaite envoyer un message alors :
- Si $statut_u \in \{NM, NP, CH\}$ et que la destination est dans son voisinage, u le transfert directement au destinataire (règle R1.a),
- Si $statut_u \in \{NM, NP\}$ et que la destination est dans son cluster, u l'envoie à son clusterhead (règle R1.b) et celui-ci le transfert au destinataire à l'étape suivante (règle R1.a).

Quand un nœud u souhaite envoyer un message et qu'il ne possède aucune entrée dans sa table de routage vers cette destination alors :
- Si $statut_u = CH$, u l'envoie à tous les nœuds de passage voisins (règle R2),
- Si $statut_u = NM$, u l'envoie uniquement à son clusterhead (règle R1.b),
- Si $statut_u = NP$, u l'envoie à son clusterhead et aux nœuds voisins n'appartenant pas à son cluster (règle R1.b) et (règle R2).

Algorithme 13 Envoi d'un message (source, destination, données, LP) depuis le nœud u

```
/* (R1) Traitement local au cluster */
Si destination ∈ N_u Alors /* Le destinataire est dans le voisinage de u (R1.a) */
   Ajouter (id_u, cl_u) à LP
   Transfert du message à destination
Sinon Si destination ∈ LC_u Alors /* Le destinataire est dans le cluster de u (R1.b) */
   Ajouter (id_u, cl_u) à LP
   Transfert du message au clusterhead cl_u
Sinon /* (R2) Routage du message */
   LE ← ∅
   Pour tout w ∈ N_u Faire
      Si (statut_w = NP) ∨ (statut_w = CH) Alors /* Envoi à tous les NP et CH */
         Si (cl_w = cl_u) Alors
            Si (statut_u = CH) ∨ (statut_w = CH) Alors
               Ajouter (id_u, cl_u) à LP
               Transfert du message à w
            Fin Si
         Sinon Si cl_w ∉ LE Alors /* Envoi uniquement aux clusters non visités */
            LE ← LE ∪ cl_w
            Ajouter (id_u, cl_u) à LP
            Transfert du message à w
         Fin Si
      Fin Si
   Fin Pour
Fin Si
```

Réception d'un message. Quand un nœud u reçoit un message alors :

- Si $statut_u \in \{CH, NP\}$ et la destination est dans le voisinage, u le transfert directement au destinataire (règle R1.a),
- Si $statut_u = NP$ et la destination est dans son cluster, u l'envoie à son cluster (règle R1.b).

Quand un nœud u reçoit un message et ne possède aucune entrée dans sa table de routage vers cette destination alors :

- Si $statut_u = NP$, u l'envoie à son clusterhead et aux clusters voisins qui ne sont pas encore visités. Pour cela, il examine d'abord la liste LP contenu dans l'entête du message (règle R1.b) et (règle R2),
- Si $statut_u = CH$, u l'envoie uniquement aux clusters voisins qui ne sont pas encore visités (règle R2).

Quand un nœud u reçoit un message et possède une entrée dans sa table de routage vers cette destination alors :

- Si $statut_u \in \{CH, NP\}$, u l'envoie directement à la passerelle correspondante.

Algorithme 14 Réception d'un message (destination, données, LP) du nœud v sur le nœud u

```
/* (R1) Traitement local au cluster */
Si destination = u Alors /* Le message est destiné à u */
  /* Traitement du message */
Sinon Si destination ∈ N_u Alors /* Le destinataire est dans le voisinage de u (R1.a) */
  Ajouter (id_u, cl_u) à LP
  Transfert du message à destination
Sinon Si destination ∈ LC_u Alors /* Le destinataire est dans le cluster de u (R1.b) */
  Ajouter (id_u, cl_u) à LP
  Transfert du message au clusterhead cl_u
Sinon /* (R2) Routage du message */
  LE ← ∅
  Pour tout w ∈ N_u Faire
    Si (w ≠ v) ∧ ((statut_w = NP) ∨ (statut_w = CH)) Alors /* Envoi à tous les NP et CH,
    sauf v */
      Si (cl_w = cl_u) Alors
        Si (statut_u = CH) ∨ (statut_w = CH) Alors
          Ajouter (id_u, cl_u) à LP
          Transfert du message à w
        Fin Si
      Sinon Si (∀x ∈ LP/cl_x ≠ cl_w) ∧ (cl_w ∉ LE) Alors /* Envoi uniquement aux clusters
      non visités */
        LE ← LE ∪ cl_w
        Ajouter (id_u, cl_u) à LP
        Transfert du message à w
      Fin Si
    Fin Si
  Fin Pour
Fin Si
```

Sur réception d'un message, l'algorithme 15 permet à chaque nœud intermédiaire de construire sa table de routage. Cet algorithme se place au niveau de la règle ($R2$) de l'algorithme 14. Chaque entrée de la table est associée à un nœud extérieur du cluster déjà rencontré. Elle contient l'identité du nœud, la distance pour atteindre le nœud et la passerelle vers ce nœud. La passerelle correspond au nœud émetteur du message. Chaque nœud qui relaie un message ajoute son identité et son cluster identité à la liste LP. Cette liste contient le parcours complet du message. Quand un nœud reçoit un message, il vérifie d'abord s'il possède une entrée vers chaque nœud de LP. S'il ne possède aucune entrée vers un nœud de LP, alors il ajoute dans sa table de routage une entrée correspondant à ce nœud. Par contre, s'il possède une entrée dans sa table de routage, il vérifie si la nouvelle entrée est meilleure que l'ancienne entrée. Si c'est le cas, il met à jour sa table avec la nouvelle entrée.

Algorithme 15 Réception d'un message (destination, données, LP) sur un nœud u

```
/* Mise à jour de la table de routage */
transfert ← vrai
Pour i allant de 0 à longueur(LP)-1 Faire
  Si LP[i]_id ∉ TR_u Alors
    TR_u ← TR_u ∪ {(LP[i]_id, longueur(LP)-i, emetteur)}
  Sinon
    e ← {(id_e, distance_e, suivant_e) ∈ TR_u / id_e = LP[i]_id}
    Si distance_e > longueur(LP)-i Alors
      TR_u ← TR_u \ {e}
      TR_u ← TR_u ∪ {(id_e, longueur(LP)-i), emetteur, }
    Sinon Si distance_e < longueur(LP)-i Alors
      transfert ← faux
    Fin Si
  Fin Si
Fin Pour
/* Transfert du message */
Si transfert = vrai Alors
  /* Suite algorithme 14 */
Fin Si
```

L'algorithme 16 permet de rechercher une entrée correspondante à une destination dans la table de routage. Cet algorithme se place au niveau de la règle ($R2$) d'algorithmes 13 et 14.

Algorithme 16 Recherche *destination* dans la table de routage de u

```
Si ∃((id_e, distance_e, suivant_e) ∈ TR_u / id_e = destination) Alors
  Ajouter (id_u, cl_u)
  Transfert du message à suivant_e
Sinon
  /* Suite algorithme 13 ou 14 */
Fin Si
```

5.9.4 Preuve

Théorème 5.1 *Un message ne passera jamais 2 fois par le même cluster*

Preuve. Nous savons que seuls les clusterheads et les nœuds de passage relayent les messages. Grâce à la variable de type liste (LP) contenu dans l'entête du message, les nœuds saurons par quels clusters le message est passé. A la réception d'un message, chaque nœud (clusterhead ou nœud de passage) examine l'entête du message. Si le cluster vers qui le nœud souhaite envoyer le message n'est pas dans la liste LP, alors le nœud relaie le message, sinon le message ne sera pas envoyé car il est déjà passé par ce cluster. Donc il n'y a pas de boucle de routage. □

Théorème 5.2 *Un message atteindra sa destination*

Preuve.

- Soit S le nœud source et D la destination tel que $cl_S = cl_D$, comme S et D sont dans le même cluster alors S envoie le message directement à D,
- Si $cl_S \neq cl_D$ alors S et D ne sont pas dans le même cluster. On aura 3 cas :
  - Si $statut(S) = NM$ alors S envoie le message directement à son clusterhead. Ensuite le clusterhead l'envoie aux clusters voisins,
  - Si $statut(S) = CH$ alors S envoie le message à tous les nœuds de passage voisins.

  A la réception d'un message, chaque nœud (clusterhead ou nœud de passage) examine la liste LP. Si le cluster vers qui le nœud souhaite envoyer le message n'est pas dans la liste alors il le relaie. Donc $\forall u, v \in V$, u peut envoyer un message à v.

□

5.10 Conclusion

La mise en place d'un protocole de routage dans les réseaux ad hoc est un problème très difficile. Les schémas de routage classiques basés sur les localisations des sites statiques sont évidemment inadaptés dans un environnement mobile. A travers notre étude des différents schémas de routage qui existent dans la littérature, nous avons vu que ces protocoles utilisent une variété de techniques afin de résoudre le problème de routage dans les réseaux ad hoc. Parmi les techniques exploitées : le principe de localisation, le concept de routage source, le principe d'inversement de liens, ...

Les solutions de routage à plat sont simples à implémentées, cependant elles ne permettent pas le passage à l'échelle pour des réseaux de grande taille. Pour implémenter une solution de routage plus efficace, *l'overhead* introduit par le protocole doit être réduit. Une façon de faire cala est de structurer le réseau avant d'implémenter un protocole de routage.

Dans ce chapitre, nous avons proposé deux solutions de routage basées sur une structure hiérarchique. Le plus gros avantage de nos solutions est qu'elles sont basées sur une structure hiérarchique, en définissant des nœuds ayant des rôles prépondérants qui sont chargés de coordonner le routage dans le réseau. Ainsi nous proposons d'utiliser un schéma de routage proactif à l'intérieur des clusters, un schéma de routage réactif sans mémoire entre les clusters et un schéma de routage hybride avec mémoire entre les cluster. Nous proposons aussi un schéma de routage basé sur un structure d'arbre couvrant. Pour le routage intra-cluster proactif, les nœuds profitent des messages de contrôle et les informations envoyées par leur clusterhead pour construire leur table des membres du cluster. Pour la partie réactive sans mémoire, nous proposons de combiner la découverte de routes et la transmission de données en une seule phase afin d'optimiser le trafic réseau. Pour la partie réactive avec mémoire, nous proposons de combiner la découverte de routes et la transmission de données en une seule phase mais parallèlement à la transmission des données les nœuds construisent leur table de routage contenant les distances et les passerelles pour atteindre les destinations déjà rencontrées. Le schéma de routage basé sur l'arbre couvrant construit également les tables de routage parallèlement à la transmission des données.

Dans le futur, il serait intéressant de proposer un mécanisme de maintenance de routes pour améliorer le taux de livraison. Nous pensons que le mécanisme de maintenance de routes peut s'appuyer sur les protocoles existants tels que DSDV, AODV, etc.
La force de ces solutions réside sur la facilité de localisation des nœuds cibles et la réduction des messages de contrôle.

Conclusion et Perspectives

Conclusion

Nous avons étudié à travers cette thèse la problématique liée au routage de données dans les réseaux ad hoc. Notre démarche consiste à organiser les réseaux en clusters afin de rendre l'utilisation du réseau plus efficace, par exemple pour le routage, la diffusion, la localisation, etc. Il nous paraissait important de structurer le réseau avant son utilisation.

Dans un premier temps, nous nous sommes d'abord intéressés au problème du clustering. Le clustering consiste à découper le réseau en clusters afin de donner au réseau une structure hiérarchique. Cette structure hiérarchique rend l'utilisation du réseau plus efficace et plus performante. Ainsi, chaque cluster est composé d'un clusterhead, des nœuds de passage et éventuellement des nœuds appelés membres. L'algorithme que nous avons développé à cet effet est un algorithme auto-stabilisant qui n'est basé que sur des connaissances locales : chaque nœud connaît que la liste de ses voisins. L'algorithme garantit qu'à partir d'une configuration quelconque, le réseau sera entièrement organisé en clusters en au plus n+2 transitions où n est le nombre de nœuds du réseau. De plus, il assure le retour automatique sans aucune intervention extérieur, à un fonctionnement normal le plus rapidement possible. Par ailleurs, nous avons notamment démontré de manière formelle la propriété auto-stabilisante de l'algorithme. À partir d'une configuration quelconque, même si un changement de topologie survient, il converge vers un état légal. L'auto-stabilisation est une caractéristique importante de notre algorithme.

Ensuite, nous nous sommes intéressés à la diffusion d'informations dans le réseau. La problématique consiste à concevoir un protocole de diffusion efficace afin de pouvoir faire de la diffusion dans le réseau. Pour cela, nous avons exploité notre structure en clusters pour proposer une utilisation supplémentaire. Ainsi, nous construisons une structure d'arbre couvrant de clusters. Ce type de structure est très largement utilisé dans la communauté réseau. Par ailleurs, aucune connaissance globale ni un quelconque GPS n'est nécessaire pour parvenir à faire un arbre couvrant de clusters. De plus notre solution, tolérante aux fautes, prend en compte les changements topologiques en choisissant une nouvelle racine et met a jour l'arbre en réexécutant l'algorithme sur la topologie modifiée. En effet, la structure d'arbre des clusters permet l'application d'un protocole de diffusion sur l'ensemble du réseau, avec un coût faible et une maintenance locale.

En fin, nous nous sommes intéressés au routage de données dans le réseau. L'objectif est de concevoir un protocole de routage basé sur la structure crée. Cependant, pour montrer que cette structure présente un intérêt, il est nécessaire d'ajouter un protocole de routage. Pour

cela, nous avons présenté trois solutions de routage sur cette structure. Le plus gros avantage de ces solutions est qu'elles basées sur une structure hiérarchique, en définissant des nœuds ayant des rôles prépondérants qui sont chargés de coordonner le routage. Ainsi, ces solutions permettent d'optimiser le routage en limitant les nœuds chargés de relayer les messages. La particularité de ces solutions est qu'elles combinent la découverte de routes et la transmission de données en une seule phase. Cela permet à la fois d'accélérer le routage et d'optimiser le trafic réseau.

La principale caractéristique des algorithmes présentés dans ce manuscrit est leur propriété auto-stabilisante. Comme les réseaux ad hoc sont des réseaux fortement dynamiques, il est important que toute conception de solutions destinées à ces réseaux soit auto-stabilisante. Par ailleurs, pour tolérer les fautes transitoires, l'approche d'auto-stabilisation permet de rendre un système distribué tolérant aux fautes.

Perspectives

Les travaux réalisés durant cette thèse nous ouvrent différentes perspectives scientifiques. Nous avons en effet proposés différents algorithmes au cours de nos travaux de recherche pour l'organisation du réseau, la diffusion d'information et le routage de données. Une première perspective de recherche vise à étudier plus en profondeur ces algorithmes afin de faire certaines optimisations. Par exemple, il serait intéressant de penser à une solution d'économie d'énergie permettant aux clusterheads de s'endormir, puisqu'ils sont responsables de coordonner le fonctionnement du réseau. Une seconde perspective serait d'étendre l'algorithme présenté au chapitre 3, pour la construction des clusters, et de proposer un algorithme qui construit des clusters à k sauts. Ceci permettrait de concevoir un protocole de routage efficace qui optimise les nœuds chargés de relayer les messages. En fin la dernière perspective s'intéresse au problème traité dans le chapitre 5. Il serait intéressant de proposer un mécanisme de maintenance de routes pour améliorer le taux de livraison. Nous pensons qu'au lieu de proposer un nouveau mécanisme de maintenance de routes, nous pouvons adapter ceux utilisés par les protocoles de routage existants. Il serait également intéressant d'étudier une solution de routage avec qualité de service.

Bibliographie

[AJRA08] S. Adabi, S. Jabbehdari, A-M. Rahmani et S. Adabi : Sbca : Score based clustering algorithm for mobile ad-hoc networks. *In ICYCS* [DBL08], pages 427–431. 27

[AjWF02] K-M. Alzoubi, P j. Wan et O. Frieder : New distributed algorithm for connected dominating set in wireless ad hoc networks. *In Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002. 29

[AP00] A-D. Amis et R. Prakash : Load-balancing clusters in wireless ad hoc networks. *In In Proceedings 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pages 25–32, 2000. 30

[APV+00] A-D. Amis, R. Prakash, T-H-P. Vuong, D-T. Huynh, T-H. P, V. Dung et T. Huynh : Max-min d-cluster formation in wireless ad hoc networks. *In in Proceedings of IEEE INFOCOM*, pages 32–41, 2000. 29

[ASK02] A.Ramalingam, S. Subramani et K.Perumalsamy : Associativity based cluster formation and cluster management in ad hoc networks. *In in Procs. of the Intl. Conf. On High Performance Computing*, 2002. 29

[ASTC96] P. Hall A-S. Tanenbaum et E. Cliffs : *Computer Networks*. NJ, 1996. 55

[AT99] G. Aggelou et R. Tafazolli : RDMAR : A bandwidth-efficient routing protocol for mobile ad hoc networks. *In Proc. 2d ACM Int'l Workshop on Wireless Mobile Multimedia (WoWMoM'99)*, pages 26–33, Seattle, Washington, USA, 1999. 93

[AWD03] M. Abolhasan, T. Wysocki et E. Dutkiewicz : A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1–22, 2003. 81, 96

[Bas99] S. Basagni : Distributed clustering for ad hoc networks. *In ISPAN* [DBL99a], pages 310–315. 21, 23

[BDDL02] N. Badache, D. Djenour, A. Derhab et T. Lemlouma : Les protocoles de routage dans les réseaux mobiles ad hoc. *Revue d'Information Scientifique et Technique (RIST)*, 12(2):77–112, 2002. 96

[BDPV99] A. Bui, A-K. Datta, F. Petit et V. Villain : State-optimal snap-stabilizing pif in tree networks (extended abstract). *In In Proceedings of the Fourth Workshop on Self-Stabilizing Systems*, pages 78–85. IEEE Computer Society Press, 1999. 10

[BG87] D-P. Bertsekas et R-G. Gallager : Distributed asynchronous bellman-ford algorithm. *In Data Networks*, chapitre 5.2.4, pages 325–333. Prentice Hall, Englewood Cliffs, 1987. 80

[BISW98] S. Basagni, C. Imrich, V-R. Syrotiuk et B-A. Woodward : A distance routing effect algorithm for mobility (dream). *In MobiCom '98 : Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 76–84, New York, NY, USA, 1998. ACM. 86

[BKL01] P. Basu, N. Khan et T-D-C. Little : A mobility based metric for clustering in mobile ad hoc networks. *In ICDCSW '01 : Proceedings of the 21st International Conference on Distributed Computing Systems*, page 413, Washington, DC, USA, 2001. IEEE Computer Society. 25

[BLB95] F. Butelle, C. Lavault et M. Bui : A uniform self-stabilizing minimum diameter tree algorithm (extended abstract). *In Proceedings of the 9th International Workshop on Distributed Algorithms*, pages 257–272, London, UK, 1995. Springer-Verlag. 65

[BR99] R-B. Bhargav et G-O. Richard : A reliable, efficient topology broadcast protocol for dynamic networks. *In INFOCOM*, pages 178–186, 1999. 87

[BS01] A. Beongku et P. Symeon : A mobility-based clustering approach to support mobility management and multicast routing in mobile ad-hoc wireless networks. *Int. J. Netw. Manag.*, 11(6):387–395, 2001. 26

[CDT02] M. Chatterjee, S-K. Das et D. Turgut : Wca : A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks*, 5:193–204, 2002. 27

[CE95] M-S. Corson et A. Ephremides : A distributed routing algorithm for mobile wireless networks. *In Wireless Networks*, volume 1, pages 61–81. Kluwer Academic Publishers, February 1995. 89

[CG98] T-W. Chen et M. Gerla : Global state routing : A new routing scheme for ad-hoc wireless networks. *In Proc. IEEE ICC'98*, pages 171–175, 1998. 83, 85

[CIS03] J. Cartigny, F. Ingelrest et D. Simplot : Rng relay subset flooding protocols in mobile ad-hoc networks. *Int. J. Found. Comput. Sci.*, 14(2):253–265, 2003. 63

[CJ03] T. Clausen et P. Jacquet : Optimized link state routing protocol (olsr), 2003. 86

[CWLG97] C-C. Chiang, H-K. Wu, W. Liu et M. Gerla : Routing in clustered multihop, mobile wireless networks with fading channel, 1997. 23

[DBL99a] *1999 International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN '99), 23-25 June 1999, Fremantle, Australia*. IEEE Computer Society, 1999. 123

[DBL99b] *2nd Workshop on Mobile Computing Systems and Applications (WMCSA '99), February 25-26, 1999, New Orleans, LA, USA*. IEEE Computer Society, 1999. 128

[DBL05] *34th International Conference on Parallel Processing (ICPP 2005), 14-17 June 2005, Oslo, Norway*. IEEE Computer Society, 2005. 129

[DBL08] *Proceedings of the 9th International Conference for Young Computer Scientists, ICYCS 2008, Zhang Jia Jie, Hunan, China, November 18-21, 2008*. IEEE Computer Society, 2008. 123

[DCWS97] R. Dube, D-R. Cynthia, K-Y. Wang et K-T. Satish : Signal stability-based adaptive routing (ssa) for ad hoc mobile networks. *In IEEE Personal Communications Magazine*, pages 36–45. IEEE, February 1997. 92

[DD92] N-F. Reynolds D. Duchamp : Measured performance of wereless lan. Technical report, Computer Science Department, Colimbia University, United States, September 1992. 15

[Dij59] E-W. Dijkstra : A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. 79

[Dij74] Edsger W. Dijkstra : Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974. 10

[DJPV98] A-K. Datta, C. Johnen, F. Petit et V. Villain : Self-stabilizing depth-first token circulation in arbitrary rooted networks. *In Distributed Computing*, page 2000. Carleton University Press, 1998. 65

[DM04] B. DERBEL et M. Mosbah : A fully distributed linear time algorithm for cluster network decomposition. *In the 16th IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS 2004*, pages 548–553, MIT, Cambridge, MA, USA, 2004. ACTA Press. 32

[DMZ06] B. Derbel, M. Mosbah et A. Zemmari : Fast distributed graph partition and application. *In Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 125–125, Washington, DC, USA, 2006. IEEE Computer Society. 32

[DMZ10] B. Derbel, M. Mosbah et A. Zemmari : Sublinear fully distributed partition with applications. *Theory of Computing Systems/Mathematical Systems Theory*, 47:368–404, 2010. 32

[DW03] F. Dai et J. Wu : Distributed dominant pruning in ad hoc networks, 2003. 62

[EWB88] A. Ephremides, J-E. Wieselthier et D-J. Baker : A design concept for reliable mobile radio networks with frequency-hopping signaling. *NASA STI/Recon Technical Report N*, 89:17772–+, septembre 1988. 22

[FHN09] O. Flauzac, B-S. Haggar et F. Nolot : Self-stabilizing clustering algorithm for ad hoc networks. *The Fifth International Conference on Wireless and Mobile Communications, ICWMC 2009*, pages 24–29, August 2009. 53

[FHN10a] O. Flauzac, B-S. Haggar et F. Nolot : Self-stabilizing tree and cluster management for dynamic networks. *10th International Conference on Innovative Internet Community*, 165:20–29, 2010. 76

[FHN10b] O. Flauzac, B-S. Haggar et F. Nolot : Tree and cluster management for manet. *The 10th Annual International Conference on New Technologies of Distributed Systems, NOTERE 2010*, 2010. 76

[FLP85] M-J. Fischer, N-A. Lynch et M-S. Paterson : Impossibility of distributed consensus with one faulty process. *Journal of the Associationof the Computing Machinery*, 32:374–382, 1985. 10

[FM02] Y. Fernandess et D. Malkhi : K-clustering in wireless ad hoc networks. *In POMC '02 : Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 31–37. ACM Press, 2002. 29

[Gar03] F-C. Gartner : A survey of self-stabilizing spanning-tree construction algorithms. Technical report, IC Swiss, Swiss, june 2003. 11

[Gau03] Vincent Gauthier : Crosslayer et QoS dans les réseaux ad hoc sans fils. Mémoire de D.E.A., Institut National des Télécommunications, Evry, France, 2003. 88, 89

[GCSRS08] A. Gallais, J. Carle, D. Simplot-Ryl et I. Stojmenovic : Localized sensor area coverage with low communication overhead. *IEEE Trans. Mob. Comput.*, 7(5):661–672, 2008. 21

[GHP02] Mario Gerla, Xiaoyan Hong et Guangyu Pei : Fisheye State Routing protocol (FSR) for ad hoc networks, 2002. IETF MANET Working Group, Internet Draft. 85

[GN+02] G.Chen, , F-G. Nocetti, , J. Solano-González et I. Stojmenovic : Connectivity-based k-hop clustering in wireless networks. *In HICSS*, page 188, 2002. 28

[GT95] M. Gerla et J. T-C. Tsai : Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995. 22

[Hag09] B-S. Haggar : Clusterisation auto-stabilisante des les réseaux ad hoc. *10èmes Journées Doctorale en Informatique et Réseaux*, pages 31–36, Fevrier 2009. 53

[HKCkW+97] C-C-C. Hsiao-Kuang, C-C. Chiang, H k. Wu, W.Liu et M. Gerla : Routing in clustered multihop, mobile wireless networks with fading channel, 1997. 84

[HR88] J-M. Hélary et M. Raynal : *Synchronisation et contrôle des systèmes et programmes répartis.* Eyrolles Ed, 1988. 8

[ICP+99] A. Iwata, C-C. Chiang, G. Pei, M. Gerla et T-W. Chen : Scalable routing strategies for ad hoc wireless networks. *IEEE Journal on Selected Areas in Communications*, 17:1369–1379, 1999. 96

[IMZ02] I.Stojmenovic, M.Seddigh et J-D. Zunic : Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):14–25, 2002. 61, 62

[JG07] G. Jayakumar et G. Gopinath : Ad hoc mobile wireless networks routing protocols - a review. *Journal of Computer Science*, 6(3):574–582, 2007. 82

[JHM07] D-A. Johnson, Y-C. Hu et D-A. Maltz : The dynamic source routing protocol (dsr) for mobile ad hoc networks for ipv4, 2007. 55, 56, 88

[JLT99] M. Jiang, J. Li et Y-C. Tay : Cluster based routing protocol(cbrp). *INTERNET-DRAFT, draft-ietf-manet-cbrp-spec-01.txt*, August 1999. 81

[JMJ01] D-B. Johnson, D-A. Maltz et J.Broch : Dsr : The dynamic source routing protocol for multihop wireless ad hoc networks, ch. 5, 2001. 55, 88

[JMQ98] P. Jacquet, P. Mühlethaler et A. Qayyum : Optimized link state routing protocol. Internet-draft, IETF MANET Working Group, November 1998. Expiration : May 1999. 86

[JN06] C. Johnen et L-H. Nguyen : Self-stabilizing weight-based clustering algorithm for ad hoc sensor networks. *In* Sotiris E. Nikoletseas et José D. P. Rolim, éditeurs : *Algorithmic Aspects of Wireless Sensor Networks, Second International Workshop, ALGOSENSORS 2006, Venice, Italy, July 15, 2006, Revised Selected Papers*, volume 4240 de *Lecture Notes in Computer Science*, pages 83–94, 2006. 20, 21, 24

[JNL99] M. Joa-Ng et I-T. Lu : A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks. *IEEE Journal on Selected Areas In Communication*, 17(8):1415–1425, 1999. 95

[KN98] Y-B. Ko et H-A. Nitin : Location-aided routing (lar) in mobile ad hoc networks. *In Mobile Computing and Networking, MOBICOM'98, October 25-30, 1998, Dallas, Texas, USA*, pages 66–75. ACM / IEEE, October 1998. 93

[KVCP97] P. Krishna, N-H. Vaidya, M. Chatterjee et D-K. Pradhan : A cluster-based approach for routing in dynamic networks. *ACM SIGCOMM Computer Communication Review*, 27:49–65, 1997. 30

[Lav86] I. Lavallée : *Algorithmique parallèele et distribuée, application à l'optimisation combinatoire*. Thèse de doctorat, Université Paris 11, 1986. 109

[Lav90] I. Lavallée : *Algorithmique parallèle et distribuée*, volume 1. Hermès, 1990. 109

[Lav00] I. Lavallée : Self-stabilizing distributed spanning tree and leader election algorithm. *ACIS Int. J Comp. Inf. Sci.*, 1:119–125, June 2000. 65

[LC00] C-H. Lin et Y-H. Chu : A clustering technique for large multihop mobile wireless networks. *In Vehicular Technology Conference Proceedings (VTC)*, Tokyo-Japan, 2000. 30

[LG97] C-R. Lin et M. Gerla : Adaptive clustering for mobile wireless networks. *IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997. 24, 28

[LHS05] N. Li, J-C. Hou et L. Sha : Design and analysis of an mst-based topology control algorithm. *IEEE Transactions on Wireless Communications*, 4(3):1195–1206, 2005. 64

[LL91] I. Lavallée et C. Lavault : Spanning tree construction for nameless networks. *In Proceedings of the 4th International Workshop on Distributed Algorithms*, WDAG '90, pages 41–56, London, UK, 1991. Springer-Verlag. 65

[LMHCH06] J-J-Y. Leu, M-H.Tsai, T-C. Chiang et Y-M. Huang : Adaptive power-aware clustering and multicasting protocol for mobile ad hoc networks. *In* Ma *et al.* [MJYT06], pages 331–340. 26

[Mal98] G. Malkin : Rip version 2. *IETF, RFC 2453*, November 1998. 80

[MB07] E. Kranakis M. Barbeau : *Principles of Ad-hoc Networking*. Hardcover, 2007. 13

[MF05] N. Mitton et E. Fleury : Efficient broadcasting in self-organizing multi-hop wireless networks. *In* Syrotiuk et Chávez [SC05], pages 192–206. 60

[Mit06] N. Mitton : *Auto-organisation des réseaux sans fil multi-sauts à grande échelle*. Thèse de doctorat, INSA de Lyon, Lyon, France, 2006. 20

[MJYT06] J. Ma, H. Jin, L-T. Yang et J-J-P. Tsai, éditeurs. *Ubiquitous Intelligence and Computing, Third International Conference, UIC 2006, Wuhan, China, September 3-6, 2006, Proceedings*, volume 4159 de *Lecture Notes in Computer Science*. Springer, 2006. 127

[Moy98] J. Moy : Ospf version 2. *IETF, RFC 2328*, April 1998. 79

[NM04] E.Fleury N. Mitton, A. Busson : Self-organization in large scale ad hoc networks. *Proc. 3rd Mediterranean ad hoc Networking Workshop (MedHocNet'04)*, 2004. 31

[NTCS99] S-Y. Ni, Y-C. Tseng, Y-S. Chen et J-P. Sheu : The broadcast storm problem in a mobile ad hoc network. *In MOBICOM*, pages 151–162, 1999. 57

[OTL04] R. Ogier, F. Templin et M. Lewis : Topology dissemination based on reverse-path forwarding (tbrpf), 2004. 87

[PB94] C-E. Perkins et P. Bhagwat : Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, 24(4):234–244, 1994. 83, 84

[PBRD03] C. Perkins, E. Belding-Royer et S. Das : Ad hoc on-demand distance vector (aodv) routing, 2003. 56, 88

[PE99] C-E. Perkins et M-B-R. Elizabeth : Ad-hoc on-demand distance vector routing. *In WMCSA* [DBL99b], pages 90–100. 56, 88

[Pel00] D. Peleg : *Distributed computing : a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. 31

[PGC00] Guangyu Pei, Mario Gerla et Tsu-Wei Chen : Fisheye state routing : A routing scheme for ad hoc wireless networks. *In Proc. IEEE Int'l Conf. on Communications, (ICC 2000)*, volume 1, pages 70–74, New Orleans, LA, USA, 2000. 85

[PGHC99] G. Pei, M. Gerla, X. Hong et C-C. Chiang : A wireless hierarchical routing protocol with group mobility. *In IEEE Wireless Communications and Networking Conference, WCNC1999, September 1999, New Orleans, LA, USA*, volume 3, pages 1538–1542. IEEE, IEEE, September 1999. 96

[QVL02] A. Qayyum, L. Viennot et A. Laouiti : Multipoint relaying for flooding broadcast messages in mobile wireless networks. *In HICSS '02 : Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 9*, page 298, Washington, DC, USA, 2002. IEEE Computer Society. 59

[Rab] C. Rabat : Dasor Home Page. http ://cosy.univ-reims.fr/∼crabat/DASOR/. 3, 46

[Rak94] H-M. Rakotoarisoa : *Simulation distribué de réseaux de files d'attente*. Thèse de doctorat, Université de Nice Sophia-Antipolis, 1994. 46

[Ray87] M. Raynal : *Systèmes répartis et réseaux : concepts, outils et algorithmes*. Eyrolles Ed, 1987. 8

[RT99] E-M. Royer et C-K. Toh : A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communication*, 6(2):46–55, 1999. 81, 90

[SC05] Violet R. Syrotiuk et Edgar Chávez, éditeurs. *Ad-Hoc, Mobile, and Wireless Networks, 4th International Conference, ADHOC-NOW 2005, Cancun, Mexico, October 6-8, 2005, Proceedings*, volume 3738 de *Lecture Notes in Computer Science*. Springer, 2005. 128

[SGLA96] M. Shree et J-J. Garcia-Luna-Aceves : An efficient routing protocol for wireless networks. *Mob. Netw. Appl.*, 1(2):183–197, 1996. 83

[SSB99] R. Sivakumar, P. Sinha et V. Bharghavan : Cedar : Core extraction distributed ad hoc routing. *EEE Journal on Selected Areas in Communications, Special Issue on Ad Hoc Networks*, 17:1454–1465, August 1999. 93

[Toh97] C-K. Toh : Associativity-based routing for ad-hoc mobile networks. *In Wireless Personal Communications Journal, Special Issue on Mobile Networking and Computing Systems*, volume 4, pages 103–139. Kluwer Academic Publishers, March 1997. 91

[Tou80] G-T. Toussaint : The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12:261–268, 1980. 63

[VC97] D-P. Vincent et M-S. Corson : A highly adaptive distributed routing algorithm for mobile wireless networks. *In INFOCOM*, pages 1405–1413, 1997. 90

[WL99] J. Wu et H. Li : A dominating-set-based routing scheme in ad hoc wireless networks. *Telecommunication Systems Journal*, 3:63–84, 1999. 61

[YC03] J. Y. Yu et P. H. Chong : 3hbac (3-hope between adjacent clusterheads) : A novel nonvel non-overlapping clustering algorithm for mobile ad hoc networks. *In IEEE Pacific Rim Conference on communication, computers and signal processing (PACRIM'03)*, 1:318–321, August 2003. 24

[YWC05] S. Yang, J. Wu et J. Cao : Connected k-hop clustering in ad hoc networks. *In ICPP* [DBL05], pages 373–380. 31

[ZM97] J-H. Zygmunt et R-P. Marc : The zone routing protocol (zrp) for ad hoc networks. Internet-draft, IETF MANET Working Group, November 1997. Expiration : May, 1998. 94

[ZRM02] X. Zou, B. Ramamurthy et S. Magliveras : Routing techniques in wireless ad hoc networks - classification and comparison. *In* N. Callaos, éditeur : *Proc. 6th World Multiconference on Systemics, Cybernetics, and Informatics (SCI 2002)*, volume IV, Orlando, Florida, USA, 2002. IIIS. 98

[Zyg97] J-H. Zygmunt : A new routing protocol for the reconfigurable wireless networks. *In Proceedings of 6th IEEE International Conference on Universal Personal Communications, IEEE ICUPC'97, October 12-16, 1997, San Diego, California, USA*, volume 2, pages 562–566. IEEE, IEEE, October 1997. 94

Auto-organisation et routage dans les réseaux mobiles ad hoc

Résumé Nos travaux se positionnent dans le cadre de l'algorithmique distribuée et plus particulièrement des réseaux ad hoc. Les réseaux ad hoc sont auto-organisés en permettant des échanges directs entre nœuds mobiles et ne reposent sur aucune infrastructure. Chaque nœud peut se déplacer librement et indépendamment des autres impliquant une modification perpétuelle de la topologie. Dans ce contexte, la probabilité que des défaillances surviennent dans le réseau est importante. Ces défaillances gênent le bon fonctionnement du réseau et peuvent même entraîner une paralysie de celui-ci. C'est pourquoi la conception de solutions pour de tels réseaux nécessitent des mécanismes de gestion de fautes. Parmi ceux-ci, l'approche d'auto-stabilisation permet à un système de gérer les fautes transitoires. Nous étendons cette approche pour répondre aux principaux problèmes liés à la mobilité des nœuds. Notre objectif est de répondre à un double besoin d'auto-organisation du réseau et d'optimisation du nombre de messages échangés. Notre approche consiste à découper le réseau en clusters afin de lui donner une structure hiérarchique. Cette dernière rend l'utilisation du réseau plus efficace et plus performante. L'algorithme que nous avons développé à cet effet est auto-stabilisant et n'est basé que sur des connaissances locales. Nous exploitons cette solution pour proposer deux utilisations efficaces : la diffusion d'informations dans le réseau et le routage. La diffusion d'informations exploite un arbre couvrant inter-clusters, construit sans surcoût, en parallèle de la clusterisation. Le routage quant à lui exploite cet arbre pour permettre à la fois d'optimiser le délai de bout en bout et le nombre de messages échangés.

Mots-clés : réseaux ad hoc, clustering, routage, diffusion, auto-stabilisation, systèmes distribués

---

Self-organizing and routing in mobile ad hoc networks

Abstract Our work relies in the domain of distributed system, more preciselly ad hoc networks. Ad hoc networks are self-organized allowing direct exchanges between mobile nodes and do not rely on any infrastruture. Each node can move freely and independently of each others involving continuous topology variability. In this context, the probability that a failure occurs in the network is high. These failures hinder the proper functioning of the network and even causes its paralysis. Therefore, designing solutions for such networks requires fault management mechanisms. Among these, a self-stabilizating approach allows the system to withstand transient faults. We extend this approach to answer the problems induced by nodes mobility. We have two main objectives : a self-organizing network and optimizing number of exchanged messages. Our approach consists in dividing the network into clusters in order to give it a hierarchical structure. This solution allows a more efficient and effective network use. The algorithm that we developed for this purpose is a self-stabilizing algorithm based only on local informations. Based on this solution, we propose two efficient use cases : Information broadcast and a routing protocol. Information broadcast uses an inter-cluster spanning tree, generated without any overhead. In the same time as the clustering process. The routing protocol uses this tree for both round trip and number of exchanged messages optimization.

Keywords : ad hoc networks, clustering, routing, broadcast, self-stabilizing, distributed system