AUTO-ORGANISATION DES RESEAUX SANS FIL MULTI-SAUTS A GRANDE ECHELLE.

Nathalie Mitton

▶ **To cite this version:**

Nathalie Mitton. AUTO-ORGANISATION DES RESEAUX SANS FIL MULTI-SAUTS A GRANDE ECHELLE.. Informatique [cs]. INSA de Lyon, 2006. Français. NNT : . tel-00599147

HAL Id: tel-00599147
https://theses.hal.science/tel-00599147

Submitted on 8 Jun 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Mémoire

présenté par

Nathalie MITTON

en vue de l'obtention du diplôme

DOCTORAT EN INFORMATIQUE ET RESEAUX

de l'INSA de Lyon

AUTO-ORGANISATION DES RÉSEAUX SANS FIL MULTI-SAUTS À GRANDE ÉCHELLE.

Soutenue le 27/03/2006.
Numéro d'ordre : 2006-ISAL-0023.

Après avis de : François BACCELLI
Catherine ROSENBERG
David SIMPLOT-RYL

Devant la commission d'examen formée de :

Bartlomiej (Bartek) BLASZCZYSZYN
Chargé de recherche à l'INRIA - TREC
Serge FDIDA
Professeur à l'Université Pierre et Marie Curie – Paris VI
Éric FLEURY (Directeur de thèse)
Professeur à l'INSA de LYON
Isabelle GUÉRIN LASSOUS (Directrice de thèse)
Chargée de recherche habilitée à l'INRIA - ARES
David SIMPLOT-RYL (Rapporteur)
Professeur à l'Université de Lille

pour les travaux effectués au Centre d'Innovations en Télécommunications & Intégration de services de l'INSA de Lyon (CITI) sous la direction du Pr. Éric Fleury et du Dr. Isabelle Guérin-Lassous.

Table des matières

Chapitre 1

Introduction

De nos jours, les gens se déplacent et communiquent de plus en plus. Ils ont désormais besoin de nouvelles technologies leur permettant de façon simple et rapide de récupérer diverses informations et de communiquer avec des personnes distantes pouvant être n'importe où dans le monde. Ces dernières années ont vu le développement technologique de nombreux composants et appareils électroniques de toute sorte pour répondre à ces nouveaux besoins. Ces appareils communicants sont de plus en plus petits, ont des capacités de calcul de plus en plus performantes et sont de plus en plus répandus. On les rencontre partout dans notre quotidien : à la maison, au bureau, dans les voitures, etc. Les accès à l'information deviennent omniprésents à travers les téléphones portables, ordinateurs portables, PDA et les technologies de communications sans fil. Cependant, l'apparition et l'expansion de ces phénomènes ont également conduit à une explosion de la complexité à tout niveau, à un point dépassant les capacités humaines à contrôler et sécuriser. L'intervention humaine doit être remplacée par une auto-gestion du système par les machines. C'est ce qu'on appelle l'*autonomic computing*. Afin d'assurer l'ubiquité des informations, les entités technologiques doivent être mises en réseau et être capables de répondre aux défis suivants :

Auto-configuration et auto-organisation. Chaque entité doit être capable de se configurer elle-même à partir d'interactions locales avec les autres entités afin de faire émerger un comportement global qui assure le bon fonctionnement du réseau. Elle doit s'adapter aux modifications de la topologie du réseau. Par exemple, les entités doivent être capables d'établir des routes pour joindre les personnes souhaitées ou accéder à une information donnée.

Auto-guérison. Lorsqu'une panne se produit, le réseau doit être capable de la localiser, de l'identifier et de l'isoler afin qu'elle ne contamine pas l'ensemble du réseau, et si possible de la corriger.

Auto-optimisation. Chaque entité doit préserver les ressources globales du réseau afin de lui assurer une durée de vie aussi longue que possible, tout en effectuant correctement les tâches qui lui sont allouées.

Auto-protection. Le réseau doit être capable d'apprendre de ses fautes et des attaques extérieures qui surviennent. Il doit pouvoir les identifier rapidement et se protéger contre elles.

Tous ces défis doivent pouvoir être adressés en tenant compte des contraintes intrinsèques des entités technologiques et des technologies de communication. L'*autonomic computing* couvre ainsi un très large domaine des réseaux informatiques, tous présentant des contraintes supplémentaires spécifiques. Au sein de ce large spectre que représente l'*autonomic computing* et qui représente un réel défi scientifique pour les années à venir, je me suis orientée lors de ma thèse vers le domaine des réseaux sans fil tels que les réseaux *ad hoc* ou les réseaux de capteurs, qui offre l'avantage d'être plus ciblé en termes d'applications, tout en englobant un grand nombre des défis scientifiques présentés ci dessus.

Les réseaux sans fil multi-sauts sont des réseaux radio mobiles sans aucune infrastructure, ce qui leur permet une implantation rapide. Ils peuvent aussi être couplés à un LAN pour étendre la couverture d'une infrastructure existante. Les entités peuvent apparaître, disparaître, se déplacer indépendamment les unes des autres. La topologie du réseau est évolutive. Les terminaux peuvent communiquer dans la limite de la portée de leur communication radio. Un schéma de communication multi-sauts est nécessaire pour permettre à deux correspondants distants de communiquer. Dans ce schéma de communication, chaque terminal peut être utilisé comme routeur pour relayer les communications d'autres terminaux. La configuration de ces routes multi-sauts est réalisée par un protocole de routage. Afin d'être efficaces, ces protocoles de routage doivent considérer les caractéristiques intrinsèques du réseau (topologie en constante évolution), des terminaux (taille mémoire et capacités de calcul limitées...) et du médium de communication (bande passante limitée, interférences...).

Il existe aujourd'hui de nombreux protocoles de routage pour de tels réseaux. Cependant, bien qu'efficaces sur des réseaux peu denses ou de petite ou moyenne taille, aucun d'eux ne peut être utilisé sur de grandes échelles car ils généreraient trop de trafic de contrôle ou nécessiteraient des tables de routage trop importantes. L'une des solutions communément proposées pour le routage sur de grandes échelles est d'introduire un routage hiérarchique en regroupant géographiquement des entités proches en *clusters* et en utilisant des schémas de routage différents au sein des clusters et entre les clusters. Une telle approche permet à chaque entité de stocker la totalité des informations de son cluster et seulement une partie des informations concernant les autres clusters et de cette façon, permet une extensibilité du réseau.

Dans ce document, je présente une solution d'utilisation de réseau sans fil multi-sauts dense. Ce document est organisé suivant les différentes étapes de cette solution : organisation du réseau, diffusion, localisation et routage. À chaque étape, les résultats obtenus ont été analysés par simulation, par des études comparatives avec des solutions existantes et, lorsque cela était possible par une analyse théorique. Le premier chapitre est la présente introduction, dans laquelle je présente les notations utilisées au cours du document et les modèles utilisés pour les études analytiques et les simulations. Dans le chapitre 2, les différentes approches proposées dans la littérature pour organiser un réseau en clusters sont présentées. Seules les méthodes qui proposent une

organisation du réseau sont mentionnées. Il existe également de nombreuses études sur l'apport de cette organisation sur la capacité du réseau comme c'est le cas de [37] mais cet aspect n'est pas l'objet de cette thèse et ces études ne sont donc pas présentées dans ce chapitre.

Le chapitre 3 décrit notre solution de *clustering* et en étudie les différentes caractéristiques. L'algorithme de *clustering* se base sur une nouvelle métrique qui permet de lisser des petits changements de topologie. Les clusters sont construits en s'adaptant à la topologie sous-jacente, sans contrainte ni paramètre fixé *a priori*. Une analyse théorique permet de dégager certaines propriétés comme une borne supérieure sur le nombre de clusters obtenus sur une surface donnée. Des simulations dégagent des caractéristiques de la structure obtenue et comparent ses performances aux structures obtenues à partir d'autres protocoles de *clustering* existants. Il en ressort que notre algorithme construit des clusters qui offrent le meilleur comportement face à la dynamicité des entités du réseau. Dans ce chapitre, nous montrons également que notre algorithme est auto-stabilisant localement, ce qui lui confère de bonnes propriétés pour le passage à l'échelle et la résistance aux fautes.

À partir des caractéristiques dégagées de la structure de clusters, nous proposons deux grandes applications indispensables à tout réseau : une diffusion efficace d'information et un processus de routage. Nous construisons donc une seule structure pour diverses applications. Le processus de diffusion est présenté et étudié dans le chapitre 4. Ce protocole de diffusion a pour avantage d'avoir deux déclinaisons : il peut permettre une diffusion globale à l'ensemble du réseau et/ou une diffusion limitée au sein d'un cluster. Il ne demande que peu d'échanges et de calculs. La diffusion ainsi générée s'avère solliciter moins d'entités que les protocoles de diffusion existants et donc consomme moins d'énergie, tout en étant plus robuste face à des cassures de liens.

Le chapitre 5 aborde le processus de routage hiérarchique appliqué à la structure. Il s'agit d'un protocole de routage indirect (*c.à.d.* effectué en deux temps) qui présente une approche originale, en considérant le schéma inverse de celui généralement utilisé dans la littérature. En effet, les caractéristiques des clusters ont montré qu'un schéma classique présenterait les mêmes problèmes d'extensibilité rencontrés dans un réseau à plat, comme nous le verrons. Ce processus de routage intègre un processus de localisation basé sur les tables de hachage distribuées et sur la mise en œuvre d'un routage adapté aux réseaux sans fil, comme le routage par intervalle. Le protocole de routage fournit des routes proches de l'optimal tout en maintenant $O(1)$ informations sur chaque entité.

Le dernier chapitre conclut ce document et donne plusieurs perspectives à l'ensemble des travaux menés au cours de cette thèse. Ce travail a été réalisé dans le cadre de ma thèse effectuée au sein du laboratoire CITI de l'INSA de Lyon et de l'équipe INRIA ARES, sous la direction d'Éric FLEURY et d'Isabelle GUÉRIN LASSOUS.

1.1 Méthodologies et notations adoptées au cours de la thèse

Un réseau sans fil multi-sauts peut être modélisé par un graphe $G = (V, E)$ où V représente l'ensemble des terminaux mobiles et E représente les liaisons radios existant entre ces stations. Dans notre approche, nous n'avons considéré que des liens radios bi-directionnels, *c.à.d* qu'un lien $e = (u, v)$ existe si les stations u et v sont à portée de communication radio l'une de l'autre.

Nous notons $dist(u, v)$ la distance euclidienne de u à v et $d(u, v)$ la distance dans le graphe (en nombre de sauts) entre les nœuds u et v. Cette distance correspond au nombre de sauts minimum que l'on doit faire pour rejoindre v depuis u. Nous notons $\Gamma_k(u)$ le k-voisinage (ou le voisinage à k sauts) du nœud u. Le k-voisinage d'un nœud est l'ensemble des nœuds à k sauts de lui : $\Gamma_k(u) = \{v \neq u \mid d(u, v) \leq k\}$. Nous notons $\delta_k(u) = |\Gamma_k(u)|$ la cardinalité de cet ensemble. On note généralement $\Gamma(u)$ pour $\Gamma_1(u)$. On remarquera que u n'appartient pas $\Gamma_k(u)$ ($\forall k > 0,\ u \notin \Gamma_k(u)$). Par définition, $\delta_1(u) = \delta(u) = |\Gamma_1(u)|$ est le degré du nœud u.

Nous désignons par $\mathcal{C}(u)$ le cluster auquel appartient le nœud u et $\mathcal{H}(u)$ son cluster-head.

Nous utilisons $e(u/\mathcal{C})$ pour dénoter l'excentricité du nœud u dans un cluster $\mathcal{C}$. L'excentricité d'un nœud est la plus grande distance entre u et tout autre nœud du même cluster $\mathcal{C}$: $e(u/\mathcal{C}) = \max_{v \in \mathcal{C}(u)}(d(u, v))$. Le diamètre d'un cluster $\mathcal{C}$, noté $D(\mathcal{C})$, est la plus grande excentricité dans ce cluster : $D(\mathcal{C}) = \max_{u \in \mathcal{C}}(e(u/\mathcal{C}))$.

1.1.1 Modèle utilisé dans les analyses stochastiques

Lors des différentes analyses théoriques que nous avons menées, nous représentons un réseau sans fil multi-sauts par un processus ponctuel de Poisson d'intensité constante λ.

Un processus de Poisson est un processus ponctuel pour lequel la disposition des points est complètement aléatoire à chaque réalisation. L'une des propriétés caractérisant le processus de Poisson est que le nombre de points dans une zone donnée est indépendant des autres points du processus. Le processus de Poisson est le processus représentant le mieux la distribution des nœuds du réseau dans l'espace. Dans cette thèse, nous n'avons considéré que des processus de Poisson homogène (intensité constante dans le plan) et isotrope (propriétés invariantes par rotation).

Pour simuler une réalisation d'un processus de Poisson homogène d'intensité λ dans un carré de surface S, on définit d'abord le nombre de points N du semis en tirant un nombre pseudo-aléatoire dans une loi de Poisson de paramètre $\lambda \times S$. Le nombre de points N prend les valeurs k avec la probabilité suivante : $\mathbb{P}(N = k) = \frac{\lambda^k}{k!} exp(-\lambda S)$. λ représente alors le nombre moyen de nœuds par unité de surface. Pour chaque point i, l'abscisse x_i et l'ordonnée y_i sont ensuite définies par un nombre pseudo-aléatoire tiré dans une loi uniforme.

Si Φ est le processus de Poisson considéré, on désigne par $\Phi(S)$ l'ensemble des points du processus Φ distribués dans la surface S. Nous considérons qu'il existe un lien entre deux points du processus u et v si $dist(u, v) \leq R$ où R est la portée de communication radio des nœuds (u et v sont voisins).

1.1.2 Modèle de simulation

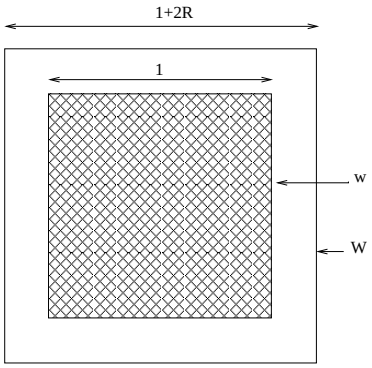


FIG. 1.1 – Seuls les nœuds de w sont considérés pour les mesures simulées mais le processus de point est distribué dans W afin d'éliminer les effets de bord.

Toutes les simulations menées lors de cette thèse suivent le même modèle. Nous avons utilisé un simulateur que nous avons développé. Ce simulateur suppose une couche MAC idéale, *c.à.d* qui ne génère aucune collision. Utiliser ce simulateur plutôt qu'un simulateur réseau qui prend en compte les collisions et caractéristiques des protocoles de niveaux inférieurs nous permet de focaliser notre étude sur le comportement des protocoles de niveau 3 uniquement, sans tenir compte des aléas des protocoles utilisés au niveau des autres couches.

Les nœuds sont déployés aléatoirement suivant un processus de Poisson dans une fenêtre carrée de $(1 + 2R) \times (1 + 2R)$ avec différentes intensités λ. On considère que deux nœuds u et v sont voisins si $dist(u, v) \leq R$ où R est la portée de communication radio des nœuds. Dans chaque cas, chaque statistique est la moyenne sur plus de 1000 simulations. Les mesures ne sont prises en compte que si l'ensemble des nœuds forment une composante connexe (aucune entité n'est isolée dans le réseau).

De façon à éliminer les effets de bords de cette fenêtre, les différentes mesures ne sont calculées que sur les nœuds se trouvant dans une fenêtre centrale w de taille 1×1 (voir Figure 1.1), l'ensemble des points du processus étant distribués dans la fenêtre W et les points de w restant impactés par les points en dehors de w. Cette technique est appelée *"minus-sampling"*. Pour une description plus détaillée, se référer à [76] page 132.

Chapitre 2

État de l'art

Un réseau *ad hoc* ne repose sur aucune infrastructure fixe. Les entités sont indépendantes les unes des autres et communiquent entre elles par radio, sans utiliser de station de base. Afin de permettre des communications entre deux stations n'étant pas à portée radio l'une de l'autre, les nœuds intermédiaires doivent relayer le message. Afin que le relais des messages soit efficace, il faut établir des routes entre les nœuds, de façon à ce que chaque entité sache vers quelle autre station envoyer le message pour qu'il puisse atteindre sa destination. C'est le rôle principal des protocoles de routage. Les protocoles de routage classiques standardisés au sein du groupe de travail MANET (Mobile Ad hoc NETwork) de l'IETF[1] se montrent efficaces sur des réseaux de petite ou moyenne taille mais passent difficilement à l'échelle [46, 71].

Il existe classiquement deux grandes familles de protocoles de routage dans la littérature et au sein du groupe MANET :

– pro-actif : les routes sont établies et maintenues en permanence sur chaque nœud. L'avantage d'un tel processus est qu'une route est disponible immédiatement quelle que soit la destination. Les inconvénients sont la taille des tables de routage à maintenir sur chaque nœud (taille en $O(n)$ si n est le nombre de nœuds dans le réseau) et le nombre de messages de contrôle à envoyer périodiquement pour maintenir à jour les routes qui ne sont pas toujours employées.
– réactif : les routes sont cherchées à la demande. L'avantage d'un tel protocole est qu'il permet d'alléger en moyenne les tables de routage et de ne pas envoyer périodiquement des messages de recherche de route. L'inconvénient est que lorsqu'une route est nécessaire, la recherche de route vers le nœud destination peut être très longue, incluant une forte latence et nécessitant une inondation du réseau.

Ainsi, avec de tels protocoles "à plat", lorsque la taille réseau grandit, le trafic de contrôle a tendance à devenir pré-dominant laissant une part congrue aux communications réelles. Cela se traduit également par une augmentation de la latence et/ou une explosion de la table de routage. Pour palier ce problème, une des solutions com-

[1] http ://www.ietf.org/html.charters/manet-charter.html

munément proposées est d'introduire un routage hiérarchique et d'organiser des nœuds en groupes aussi nommés *clusters*.

Le *clustering* consiste en un découpage virtuel du réseau en groupes de nœuds proches géographiquement. Ces groupes sont appelés *cluster*s. Ils sont généralement identifiés par un nœud particulier, un chef de groupe aussi nommé *cluster-head*. Dans la plupart des algorithmes de *clustering*, les clusters sont construits à partir d'une métrique particulière qui permet d'assigner un chef à chaque nœud ; le cluster étant alors constitué du *cluster-head* et de tous les nœuds qui lui sont rattachés. L'idée initiale du routage hiérarchique est de permettre à chaque entité de stocker la totalité des informations de son cluster et seulement une partie des informations concernant les autres clusters. Cela minimise la taille des tables de routage et la quantité de trafic généré.

Outre le fait de rendre le routage plus efficace, le *clustering* présente également d'autres avantages. Il peut faciliter le partage des ressources et/ou la synchronisation au sein d'un cluster et permettre une ré-utilisation spatiale des fréquences radio pour minimiser les interférences [50]. Plus important encore, l'organisation d'un réseau apporte aussi plus de stabilité [61].

De nombreuses solutions de *clustering* ont été proposées. La majorité d'entre elles proposent l'utilisation d'une métrique qui permet aux nœuds de se choisir un chef. Cette métrique peut être par exemple l'identifiant ou le degré des nœuds, une valeur de mobilité des nœuds ou encore une somme pondérée de tous ces éléments. D'autres solutions cherchent dans un premier temps à déterminer un ensemble dominant connecté sur lequel les clusters sont bâtis. Une grande partie des solutions de *clustering* construisent des clusters à 1 saut (dits 1-clusters), *c.à.d* des clusters où chaque nœud est à un saut de son chef de cluster. Les protocoles donnant naissance à des k-clusters (clusters où chaque nœud est à au plus k sauts de son cluster-head) sont plus récents et plus rares.

Dans ce chapitre, nous dressons un état de l'art qui permet de passer en revue les principaux types de solutions proposées dans la littérature pour organiser un réseau *ad hoc* en clusters.

2.1 Clusters à 1 saut

De nombreux algorithmes de *clustering* produisent des clusters à 1 saut. L'un des algorithmes les plus anciens est "l'algorithme du plus petit ID" ou LCA, proposé initialement par Ephremides, Wieselthier et Baker dans [28]. Chaque nœud se désigne ou non cluster-head en se basant sur son identifiant et celui de ses voisins. Un nœud peut avoir trois statuts différents : cluster-head, passerelle ou nœud ordinaire. À l'origine, tous ont un statut de nœud ordinaire. Si un nœud u a le plus petit identifiant parmi les nœuds de son voisinage, il se déclare cluster-head. Sinon, il attend que tous ses voisins ayant un identifiant plus petit que le sien ait déclaré leur statut. Si au moins l'un d'eux s'est déclaré chef, u déclare à son voisinage son statut de nœud ordinaire. u appartient alors à chacun des clusters de ses voisins s'étant déclaré chef. Si tous les voisins de u ayant un identifiant plus petit que celui de u se sont déclarés nœuds ordinaires (car ils

se sont attachés à un autre de leur voisin de plus petit ID), u se déclare cluster-head. Une fois que chaque nœud a déclaré son statut de nœud ordinaire ou de cluster-head, si un nœud entend parmi ses voisins plus d'un cluster-head, il se déclare passerelle. Le protocole LCA est notamment utilisé par le routage CBRP (Cluster Based Routing Protocol) [42], pour la formation des clusters.

Par la suite, avec le protocole HCC (High Connectivity Clustering), Gerla et Tsai [36] ont cherché à apporter plus de stabilité à la structure de clusters formés par le LCA, en utilisant le degré des nœuds plutôt que leur identifiant. Le nœud ayant le plus fort degré dans son voisinage se déclare cluster-head. Si deux voisins ont le même degré, c'est celui de plus petit identifiant qui prend sa décision le premier. L'idée est que des nœuds de fort degré sont de bons candidats pour être cluster-heads car ils couvrent un grand nombre de nœuds et le nombre de clusters résultant en sera réduit. Par ailleurs, l'identifiant d'un nœud étant unique, un nœud de faible ID aura tendance à rester cluster-head longtemps, malgré la mobilité des nœuds. Néanmoins, si ce nœud est très mobile, il détruira constamment la structure.

Ainsi, ces protocoles construisent des clusters à 1 saut qui se recouvrent (les passerelles appartiennent à plusieurs clusters). Cette structure a été proposée pour acheminer les messages de contrôle et de routage où seuls les cluster-heads et les passerelles agissent. Leur maintenance s'avère coûteuse car le mouvement d'un nœud peut engendrer des réactions en chaîne et nécessiter une reconstruction totale de la structure. C'est pourquoi les auteurs de [24] ont proposé "Least Cluster Change" (LCC). LCC ajoute une étape de maintenance des clusters formés avec le LCA ou le HCC. Les clusters ne sont reconstruits que si deux cluster-heads se retrouvent voisins (le nœud de plus faible degré et/ou de plus fort ID suivant le cas abandonne le rôle de cluster-head) ou si un nœud ordinaire n'a plus aucun cluster-head dans son voisinage (il relance le processus de clustering). De cette façon, LCC améliore la stabilité de la structure. Cependant, les réactions en chaîne de re-construction ne sont que limitées et ne sont pas complètement supprimées du fait qu'un seul nœud peut re-lancer la procédure de *clustering* s'il n'a plus aucun cluster-head dans son voisinage.

Le protocole MOBIC [13], autre protocole de *clustering* à 1 saut, applique le même algorithme que LCA et HCC mais utilise une métrique basée sur la mobilité plutôt que le degré ou l'identifiant des nœuds. Cette métrique cherche à caractériser la mobilité relative d'un nœud. L'idée est qu'un nœud peu mobile est un bon candidat pour être cluster-head car stable. Pour calculer sa mobilité relative, un nœud mesure le niveau de signal qui l'unit à chacun de ses voisins. La mobilité d'un nœud u est calculée à partir des rapports entre ce niveau de signal et celui mesuré à l'étape précédente pour chaque voisin de u, l'atténuation du signal étant dépendante de la distance séparant les nœuds. Le nœud dont la mobilité est la plus faible dans son voisinage devient cluster-head. Les auteurs de MOBIC utilise l'algorithme LCC pour la maintenance de leur structure en ajoutant une règle supplémentaire : si deux cluster-heads u et v arrivent dans le voisinage l'un de l'autre, le cluster-head v de plus fort identifiant n'abandonne son rôle de cluster-head que si u fait toujours partie de ses voisins après une certaine période de temps. Cela permet de ne pas reconstruire la structure si deux cluster-heads ne se retrouvent voisins que pour une courte période. La mobilité des nœuds n'est plus reconsidérée par la suite à moins d'avoir à reconstruire toute la structure. Cependant, les

inconvénients du LCC ne sont pas éliminés. Bien que la prise en compte de la mobilité des nœuds semble intéressante pour déterminer les cluster-heads, cette méthode est un peu complexe et nécessite que les nœuds soient en mesure d'estimer les puissances de signal. De plus, elle ne considère pas certains phénomènes physiques qui provoquent des atténuations hétérogènes du signal.

Plutôt que d'utiliser l'identifiant ou le degré des nœuds, d'autres protocoles de *clustering* utilisent une somme pondérée de plusieurs métriques. Cette catégorie d'algorithmes vise à élire le cluster-head le plus adapté à une topologie pour une utilisation donnée. Par exemple, dans un réseau de senseurs où l'énergie est un facteur important, le paramètre d'énergie résiduelle peut obtenir un poids plus élevé dans la somme pondérée de la métrique résultante. WCA [20] est un protocole utilisant une somme pondérée de quatre critères : la différence de degré D_v, la somme des distances avec les voisins P_v, la vitesse relative moyenne M_v et le temps de service en tant que cluster-head. Pour un nœud v, la différence de degré D_v est la différence entre le degré de v et une constante M représentant le nombre de nœuds qu'un cluster-head peut servir. Cependant, les auteurs n'explicitent pas le moyen de déterminer M. La mobilité relative M_v est obtenue comme dans MOBIC. Les distances P_v entre v et ses voisins sont calculées à l'aide d'un GPS. L'élection se fait en se basant là encore sur l'algorithme de LCA, le nœud dont la somme pondérée de ces critères est la plus petite devenant cluster-head. Les clusters sont ensuite maintenus sans plus reconsidérer la métrique pondérée. Le processus de *clustering* est relancé quand un nœud arrive dans une zone couverte par aucun cluster-head, ceci pouvant entraîner des réactions de reconstruction en chaîne comme dans les algorithmes précédents.

Ainsi, plusieurs méthodes de *clustering* à 1 saut se basent sur l'algorithme du LCA et changent juste le critère de décision. C'est pourquoi Basagni, dans [12] reprend l'algorithme de LCA en donnant comme critère un poids générique que chacun définit comme il le souhaite. Il en étudie alors théoriquement les différentes propriétés.

Toutes les méthodes de *clustering* mentionnées jusqu'à maintenant produisent des clusters recouvrants, *c.à.d.* une structure dans laquelle un nœud peut appartenir à plusieurs clusters. Leur inconvénient majeur est que le mouvement d'un nœud peut provoquer la re-construction d'un cluster, qui, par réaction en chaîne, provoque le re-construction de la structure entière. Afin d'éviter cela, d'autres protocoles de *clustering* ont été proposés, produisant des clusters non re-couvrants : un nœud appartient à exactement un cluster.

Dans 3HBAC [82], les auteurs proposent un protocole qui impose trois sauts entre deux cluster-heads. Le nœud ayant le plus fort degré dans son voisinage se déclare cluster-head. Ses voisins s'attachent à lui et se déclarent "nœuds membres". Les nœuds voisins de ces nœuds membres et non voisins d'un cluster-head se déclarent "unspecified" et ne peuvent plus être cluster-head. Lorsque deux cluster-heads se retrouvent dans le voisinage l'un de l'autre, celui de plus grand identifiant abandonne son rôle de cluster-head et devient un nœud membre. Ses voisins deviennent soit membres (s'ils sont voisins du cluster-head) soit non spécifiés. Les réactions en chaîne de re-construction sont ainsi évitées.

Dans "Adaptive Clustering" [50], les auteurs n'utilisent le statut de cluster-head que

pour la formation des clusters. Une fois les clusters formés, la notion de cluster-head disparaît, chaque nœud du cluster tenant alors le même rôle. La motivation des auteurs est que les cluster-heads peuvent devenir des goulots d'étranglement par la suite, sources de perte de trafic et saturation de bande passante. De plus, les cluster-heads seraient appelés à dépenser leur énergie plus vite que les autres nœuds. Pour construire de tels clusters, chaque nœud maintient un ensemble Γ qui initialement contient les identifiants de tous ses 1-voisins. Un nœud n'est autorisé à diffuser son statut (cluster-head, membre, non spécifié) que s'il possède un identifiant plus petit que les nœuds de Γ. Il ne se déclare cluster-head que s'il a un identifiant plus petit que tous les nœuds de son ensemble Γ. Sur réception du statut d'un nœud u, les voisins de u suppriment u de leur ensemble Γ. Si u a annoncé qu'il était cluster-head, ses voisins s'attachent à lui s'ils n'étaient encore membres d'aucun cluster ou si le cluster-head auquel ils s'étaient attachés avait un identifiant plus grand que u. Le processus s'arrête lorsque l'ensemble Γ de chaque nœud est vide. Comme le rôle de cluster-head disparaît une fois les clusters formés, la maintenance de la structure est un peu différente que dans les cas précédents. Chaque nœud doit connaître son voisinage à deux sauts. De cette façon, il sait si les membres de son cluster restent 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 fort degré dans le cluster reste dans le cluster. L'autre doit se rattacher à un autre cluster. Bien que n'utilisant pas la notion de cluster-head, la maintenance de cet algorithme maintient le nœud de plus fort degré au centre du cluster, ce qui peut revenir au même que de l'élire comme cluster-head. Le protocole de maintenance de l'"Adaptive Clustering" a ensuite été repris par les auteurs de [44] qui se proposent de l'appliquer au LCA.

Tous les algorithmes décrits jusqu'à présent peuvent être qualifiés de protocoles de "*clustering* actif", *c.à.d.* que des messages de contrôle sont envoyés dans le but de construire et maintenir les clusters. À l'opposé, les auteurs de [47] proposent un protocole de "*clustering* passif", *c.à.d.* qu'ils n'utilisent aucun message dédié à la construction des clusters. Les clusters ne sont créés que lorsque nécessaires, *c.à.d.* lorsqu'un nœud a une information à diffuser. Le protocole de *clustering* passif utilise alors ces messages d'information pour construire les clusters, en ajoutant des champs aux paquets d'information. Un nœud a quatre statuts possibles : cluster-head, passerelle, nœud ordinaire et non défini. Par défaut, le statut des nœuds est non défini. Seul un nœud ayant un statut non défini peut devenir cluster-head. Si un tel nœud a un message à envoyer, il se déclare cluster-head et diffuse son statut en l'ajoutant à l'information qu'il devait envoyer. Les nœuds voisins d'un cluster-head deviennent des nœuds ordinaires, les nœuds voisins de plusieurs cluster-heads deviennent des passerelles. Les nœuds ordinaires ne relaient pas les messages de diffusion. Aucun message n'étant dédié à la maintenance de la structure, les passerelles et les nœuds ordinaires activent des compteurs lorsqu'ils reçoivent des nouvelles de leur(s) cluster-head(s). S'ils restent sans nouvelle d'eux le temps que leur compteur expire, les nœuds ordinaires reprennent un statut non défini et les passerelles prennent le statut de nœud ordinaire ou non défini suivant le nombre de cluster-heads qu'elles entendent encore.

Comme nous venons de le voir, il existe de nombreux protocoles de *clustering* à 1 saut. Les solutions les plus anciennes proposaient des clusters recouvrants. Ce type de

clusters permet principalement de bâtir un ensemble dominant connecté sur le réseau (constitué des cluster-heads et des passerelles) pour pouvoir diffuser une information (principalement pour le routage) sur le réseau sans solliciter tous les nœuds. Puis d'autres études ont donné des clusters non-recouvrants, plus robustes face à la mobilité des nœuds. Ce type de clusters permet également d'autres applications comme la réutilisation spatiale de fréquences ou de codes (les nœuds de deux clusters non voisins peuvent utiliser la même fréquence). Puis, des propositions plus récentes sont apparues permettant la construction de clusters à k sauts, encore plus robustes et permettant de nouvelles applications comme l'application de zones de services ou de protocole de routage hiérarchique.

2.2 Clusters à k sauts

La méthode la plus répandue pour la construction de clusters à k sauts est une extension des algorithmes de *clustering* à 1 saut. Par exemple, les auteurs de [23] généralisent l'algorithme de Lin et Gerla [50]. Leur algorithme suppose que chaque nœud connaît ses voisins situés jusqu'à k sauts de lui. Le nœud ayant le plus petit identifiant parmi les nœuds à au plus k sauts de lui, diffuse son statut de cluster-head à ses k-voisins. Lorsque tous les nœuds de son k-voisinage ayant un plus identifiant que lui ont diffusé leur décision d'être chef de cluster ou de s'attacher à un autre chef, le nœud u peut prendre sa propre décision de s'attacher au nœud de son k-voisinage de plus petit identifiant s'étant déclaré chef de cluster s'il existe, ou de créer son propre cluster sinon. De la même façon que pour les clusters à 1 sauts, ce même algorithme est utilisé en utilisant différentes métriques. Dans le même papier [23], les auteurs proposent également d'utiliser le k-degré (δ_k) des nœuds (nombre de voisins à au plus k sauts) pour déterminer le cluster-head : le nœud de plus fort k-degré et de plus petit identifiant en cas d'égalité est promu chef de cluster. Les clusters résultants sont des k-clusters (chaque nœud est à au plus k sauts de son chef) recouvrants (un nœud peut appartenir à plusieurs clusters). Deux chefs sont éloignés d'au moins $k + 1$ sauts. Cependant, nous retrouvons les mêmes inconvénients que pour les algorithmes de clusters à 1 saut, à savoir qu'un petit changement de nœuds peut engendrer une reconstruction complète de la structure.

Les auteurs de [67] introduisent une métrique qu'ils appellent "associativité" qui se veut représenter la stabilité relative des nœuds dans leur voisinage. Pour chaque nœud, l'associativité comptabilise le temps que chacun des nœuds de son voisinage reste effectivement dans son voisinage et en fait la somme sur chaque voisin. À chaque période de temps, un nœud u considère quels sont ses voisins actuels déjà présents lors de la période précédente et ajoute $+1$ à la valeur associée à chacun d'eux. Si un voisin a disparu, la valeur qui lui était associée passe à 0, si un autre apparaît, il prend la valeur 1. À chaque période de temps, l'associativité de u est la somme des valeurs associées à chacun de ses voisins. Cette valeur prend donc en compte la stabilité de u (si u est relativement stable dans son voisinage, il aura une forte associativité) et le degré des nœuds, cette valeur n'étant pas normalisée. L'algorithme de formation des clusters est le suivant. Un nœud considère les nœuds de son k-voisinage ayant un degré supérieur à une

valeur seuil et élit parmi eux celui ayant la plus forte associativité. Le plus fort degré et le plus faible identifiant sont ensuite utilisés pour rompre les égalités. Les clusters résultants sont également des k-clusters recouvrants mais qui visent à être plus stables dans le temps et dans l'espace que ceux se basant sur le simple degré ou identifiant.

Dans [51], Lin et Chu proposent une approche basée sur aucune métrique particulière. Lorsqu'un nœud u arrive dans le réseau, il est en phase "d'initialisation". Il demande alors à ses voisins s'ils sont comme lui en phase d'initialisation ou s'ils ont un cluster-head et dans ce cas, à quelle distance ce cluster-head se situe-t-il. Si tous les voisins de u sont en phase d'initialisation, u s'élit chef de cluster et diffuse cette information. Tous les r-voisins de u qui n'ont aucun autre chef plus proche que u s'attache au cluster de u. Sinon, u s'attache au cluster de son voisin dont le chef est le plus proche et à au plus r sauts de lui. Si tous les cluster-heads des clusters de ses voisins sont à plus de r sauts de u, u se déclare chef de cluster et rallie à son cluster tous ses voisins à moins de r sauts dont le chef est plus éloigné que u. Si deux cluster-heads se retrouvent à moins de D sauts l'un de l'autre, $D < r$, le chef de cluster de plus faible identifiant doit céder son rôle et tous les membres de son cluster doivent se trouver un autre chef. Cette méthode de *clustering* est intéressante dans la mesure où elle produit des r-clusters non recouvrants où les chefs sont éloignés de au moins D sauts. Cela assure une certaine stabilité à la structure. Cependant, l'abandon du rôle de cluster-head par un nœud peut engendrer de fortes réactions en chaîne.

Une approche plus originale est celle proposée par Fernandess et Malkhi dans [32]. Leur algorithme se décompose en deux phases. La première étape consiste à trouver un arbre couvrant du réseau basé sur un ensemble dominant connecté de cardinalité minimale (MCDS). Les auteurs proposent d'utiliser l'algorithme de [2] pour construire le MCDS mais précisent que n'importe quelle méthode peut être utilisée. La seconde phase de l'algorithme consiste en une partition de l'arbre couvrant en $2k$-sous-arbres, un $2k$-sous-arbre étant un arbre de diamètre au plus $2k$ sauts. Chaque sous-arbre consiste en un k-cluster. Cependant, une telle approche a une complexité temporelle et une complexité en messages en $O(n)$ (n étant le nombre de nœuds dans le réseau) et est par conséquent difficilement extensible. De plus, les auteurs n'abordent pas la maintenance d'une telle construction, qui ne semble pas triviale.

Les auteurs de Max-Min d-cluster [4] utilisent l'identifiant des nœuds pour construire des k-clusters non recouvrants. Cependant, leur algorithme est un peu plus complexe que ceux vus jusqu'à maintenant. Il se décompose en trois phases. Lors de la première phase, chaque nœud collecte l'identifiant de ses voisins jusqu'à d sauts et en garde le plus grand qu'il diffuse de nouveau à d sauts lors de la seconde phase. Chaque nœud garde alors le plus petit des identifiants qu'il reçoit lors de cette deuxième phase (le plus petit parmi les plus grands). La troisième étape consiste au choix du cluster-head basé sur les identifiants collectés lors des deux phases précédentes. Si un nœud u a vu passer son propre identifiant lors de la deuxième phase, il devient chef de cluster. Sinon, si u a vu passer un identifiant durant chacune des phases 1 et 2, il élit le nœud portant cet identifiant comme chef. Sinon, u élit comme chef le nœud de plus grand identifiant dans son d voisinage. La structure résultante s'avère robuste, cependant la latence induite par l'algorithme est non négligeable.

Dans [3], les mêmes auteurs introduisent une notion d'identifiant virtuel. Le but est d'apporter une certaine équité entre les nœuds et d'éviter qu'un même nœud soit trop longtemps cluster-head et épuise ainsi ses ressources, tout en assurant qu'il le reste suffisamment longtemps pour apporter une stabilité à la structure. Les nœuds prennent le rôle de cluster-head tour à tour. Initialement, l'identifiant virtuel d'un nœud est égal à son propre identifiant. À chaque période de temps, chaque nœud non cluster-head incrémente de 1 son identifiant virtuel jusqu'à atteindre un maximum MAX_{VID}. Le nœud ayant l'identifiant virtuel le plus fort parmi ses k-voisins devient le *cluster-head*. En cas de conflits, c'est le nœud qui a le moins opéré en tant que chef qui devient cluster-head (et de plus fort identifiant *normal* si toujours égalité). Un nœud qui devient cluster-head prend ajoute à son ancienne valeur d'identifiant virtuelle MAX_{VID} de façon à assurer qu'il conserve le plus fort identifiant virtuel et reste cluster-head. Un nœud reste cluster-head pendant une période de temps $\Delta(t)$ au bout de laquelle il passe son identifiant virtuel à 0 et abandonne son rôle de chef. Lorsque deux chefs entrent dans le voisinage l'un de l'autre, celui de plus faible identifiant virtuel abandonne son rôle. Dans le même papier, les auteurs proposent également une construction où l'identifiant virtuel de base serait le degré des nœuds. Cet algorithme permet donc la formation de k-clusters en assurant une certaine stabilité de la structure. Néanmoins, elle nécessite une synchronisation des nœuds afin que chacun se base sur la même période de temps pour incrémenter son identifiant virtuel et surtout pour comptabiliser la période durant laquelle il est cluster-head. Or, une synchronisation dans de tels réseaux est non triviale et nécessite beaucoup de messages.

Les auteurs de [45] proposent un autre type d'algorithme, formant cette fois des clusters sans chef de cluster. Pour cela, chaque nœud nécessite également la connaissance de son k-voisinage. Un cluster est formé par un ensemble de nœuds tel qu'il existe entre deux nœuds de cet ensemble un chemin d'au plus k-sauts. Si $k = 1$, chaque cluster est une clique. Un nœud appartenant à plusieurs clusters est dit *nœud frontière*. Les clusters sont donc recouvrants. Malheureusement, cet algorithme implique beaucoup de messages de contrôle, de maintenance et de données à gérer par les nœuds.

Les auteurs de DDR [59] proposent également une structure sans cluster-head. Contrairement à la plupart des algorithmes de formation de k-clusters, les nœuds ne nécessitent que de la connaissance de leur 1-voisinage. La formation des clusters se base sur la construction d'un arbre. Chaque nœud choisit comme père son voisin de plus faible identifiant. Il existe alors exactement une arête sortante par nœud. Cela conduit à la formation d'un arbre. Tous les nœuds du même arbre appartiennent au même cluster. Le diamètre de tels clusters n'est pas fixé a priori et s'adapte automatiquement à la topologie sous-jacente. Cet algorithme a été ensuite repris par Baccelli [7] en y ajoutant la notion de cluster-head et en contrôlant la taille des clusters. Pour cela, un nœud a le droit de se choisir comme père s'il a le plus fort identifiant dans son 1-voisinage. Il existe alors des nœuds sans arête sortante qui deviennent des cluster-heads. Ces cluster-heads ont alors la possibilité de borner la hauteur des arbres à d sauts en diffusant l'information le long des branches de l'arbre. Si la branche est trop longue, le nœud se trouvant à $d+1$ sauts de son cluster-head doit s'attacher à un autre père (et donc casser la branche).

D'autres algorithmes comme ceux proposés dans [39, 60] ne proposent qu'une solution

de maintenance. Par exemple, les auteurs de [60] proposent de maintenir un certain nombre de nœuds dans un cluster, qui dépendrait du nombre d'entités que le cluster-head est en mesure de gérer. L'idée est de maintenir en permanence le nombre de nœuds entre deux seuils. Si un cluster est trop petit, le chef de cluster doit élire parmi ses clusters voisins celui le plus adapté pour une fusion, c'est-à-dire celui dont le nombre de nœuds permet la fusion des deux clusters. Si aucun ne correspond, le chef de cluster doit déterminer un cluster qui peut lui céder des entités pour un meilleur équilibrage du nombre de nœuds. Si les clusters sont trop gros, le chef doit élire parmi ses membres un autre cluster-head et scinder son cluster en deux. Il reste cluster-head d'un cluster résultant tandis que le nœud qu'il a élu devient chef du second cluster. Cette méthode est cependant très coûteuse en calculs, latence et messages et supporte mal le passage à l'échelle du réseau.

2.3 Clusters hiérarchiques

Il existe également des propositions de structures hiérarchiques à plusieurs niveaux, c'est-à-dire où les clusters sont ensuite regroupés en d'autres clusters de niveaux supérieurs et ainsi de suite. Bien que la majorité des algorithmes vus jusqu'à maintenant peuvent être appliqués récursivement sur les clusters pour former des clusters de niveau supérieur, ils n'ont pas été écrits dans ce but contrairement aux exemples que nous énonçons ici.

Dans [11], Banerjee et Khuller se basent sur un arbre couvrant, construit grâce à un parcours en largeur, pour la construction de k-clusters. Les clusters sont formés par branche, en fusionnant récursivement deux sous-arbres de l'arbre couvrant jusqu'à obtenir une taille correcte. Le processus est alors ré-itéré jusqu'à obtenir un certain nombre de niveaux.

Dans [5], les auteurs cherchent à combiner les partitions physiques et logiques des nœuds ainsi que leur mobilité. Pour cela, ils utilisent un GPS. Les auteurs supposent que les nœuds répondent à un modèle de mobilité de groupe. L'algorithme consiste ensuite à regrouper en un même cluster les nœuds proches géographiquement et qui se déplacent à un vitesse semblable dans une même direction. Le processus est ensuite ré-itéré jusqu'à obtenir le nombre de niveaux voulu.

La structure de cellules hiérarchiques de SAFARI [69] est basée sur une auto-sélection des nœuds en tant que *drums* (cluster-heads). Le nombre de niveaux hiérarchiques s'établit automatiquement en fonction de la topologie sous-jacente des nœuds. Les clusters de niveau i sont groupés en clusters de niveau $i+1$ et ainsi de suite, les simples nœuds étant considérés comme des cellules de niveau 0. Chaque cluster-head de niveau i se choisit un cluster-head de niveau $i+1$. Tous les cluster-heads de niveau i ayant choisi le même cluster-head de niveau $i+1$ appartiennent au même cluster de niveau $i+1$. Un cluster-head u de niveau i décide de monter ou descendre son niveau en fonction du nombre de cluster-heads de niveau $i+1$ et $i-1$ qui existent à une certaine distance. S'il n'existe aucun cluster-head de niveau supérieur à une distance plus petite que D_i (D_i constante dépendant du niveau i du cluster-head) de u, u décide

d'augmenter son niveau. Si deux cluster-heads de même niveau sont à moins de $h \times D_i$ ($0 < h < 1$, facteur d'hystéresis) sauts, le cluster-head de plus grand identifiant descend son niveau. Un cluster-head de niveau i est également cluster-head de tout niveau j tel $0 < j < i$. Cet algorithme construit des k-clusters hiérarchiques, où k dépend du niveau du nœud i : $k = D_i$. D_1 doit être fixé. À partir de là, D_i dépendant de D_{i-1}, le rayon des clusters de chaque niveau est fixé. Cette structure hiérarchique peut cependant n'être utilisée que dans un cadre précis de routage, proposé par les auteurs. Nous verrons cette utilisation plus en détail dans le chapitre 5.

2.4 Conclusion

Ainsi, il existe de nombreux protocoles de *clustering* dans la littérature. Tous cependant ne sont pas adaptés à une extension du réseau comme nous avons pu le constater. En effet, des clusters à 1 saut ne peuvent pas être utilisés dans ce cadre du fait du nombre de clusters qu'ils généreraient sur de la larges échelles et du fait que le moindre changement à l'échelle d'un nœud provoquerait une reconstruction de la structure. En effet, si le réseau compte beaucoup d'entités, ces changements peuvent être fréquents et minimes à l'échelle du réseau. Les clusters à k sauts sont moins développés. Beaucoup s'inspirent des protocoles de *clustering* à 1 saut et en gardent les inconvénients. Dans ma thèse, j'ai proposé un nouvel algorithme de *clustering* à k sauts pouvant s'adapter aux petites modifications du réseau. Cet algorithme prend note des inconvénients des protocoles existants et tente de les éviter, comme nous le verrons dès le chapitre suivant.

Chapitre 3

Algorithme de clustering, stable et robuste

3.1 Introduction

Notre principal objectif est de proposer un moyen d'utiliser des réseaux sans fil très denses. Comme nous l'avons vu, l'une des solutions possibles est d'introduire une hiérarchie dans le réseau en construisant des clusters. Afin de permettre une extensibilité totale et ne pas avoir à reconstruire les clusters après chaque mouvement individuel d'un nœud, nous avons cherché à construire des clusters qui n'aient aucun paramètre fixé à l'avance, qu'il s'agisse du rayon, du diamètre ou de nombre de nœuds par cluster. Ces paramètres doivent s'adapter d'eux-mêmes à la topologie du réseau, qui évolue au cours du temps. De plus, l'heuristique se doit d'être distribuée et asynchrone tout en minimisant le nombre d'informations à échanger. Notre algorithme n'utilise que des messages de type "PAQUET HELLO" comme ceux utilisés dans OLSR [25] afin de découvrir le 2-voisinage d'un nœud. Les clusters formés doivent être stables (les cluster-heads doivent conserver ce statut suffisamment longtemps pour limiter le trafic de contrôle nécessaire à la reconstruction des clusters) tout en s'adaptant aux changements de la topologie sous-jacente. Enfin, afin d'améliorer la stabilité de la structure, étant donné que les nœuds trop mobiles pour initier une communication n'ont aucun besoin de la structure, ils ne participent pas à la phase de construction et restent des nœuds indépendants. Dans le cas contraire, de par leur mobilité, ils pourraient obliger le réseau à re-construire inutilement les clusters.

Comme mentionné dans le chapitre 2, différentes métriques ont été utilisées pour le choix des cluster-heads dans les algorithmes de *clustering*. L'identifiant des nœuds étant immuable, il permet de conserver les chefs de cluster très longtemps. Cependant de tels clusters sont indépendants de la topologie sous-jacente et ne sont pas toujours adaptés. Le degré des nœuds s'avère l'une des métriques les plus adaptées, l'idée étant

qu'un chef de fort degré permet de couvrir un grand nombre de nœuds, ce qui permet d'en minimiser le nombre. Cependant, un mouvement individuel d'un nœud dans des clusters basés sur cette métrique peut conduire à une ré-organisation complète du réseau, alors que la structure globale du réseau reste inchangée.

Basés sur cette constatation, nous avons introduit une nouvelle métrique, que nous avons appelée *densité*. L'idée est que, si un petit changement de topologie intervient dans le voisinage d'un nœud, son degré δ peut changer alors que globalement son voisinage est conservé. Notre métrique est une *densité* de liens et cherche à lisser les petits changements de topologie qui interviennent au niveau individuel d'un nœud, tout en permettant aux clusters de s'adapter à la topologie sous-jacente.

3.2 La métrique de densité

La k-densité d'un nœud, notée $\rho_k(u)$, est le ratio du nombre de liens par le nombre de nœuds dans le k-voisinage d'un nœud.

Définition 1 (densité) *La k-densité d'un nœud $u \in V$ est*

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

La 1-densité (également notée $\rho(u)$) est donc le rapport entre le nombre de liens entre u et ses voisins plus le nombre de liens entre les voisins de u et le nombre de ses voisins (par définition, son degré).

Afin d'illustrer cette métrique, prenons l'exemple représenté sur la Figure 3.1. Considérons le nœud p et calculons sa 1-densité $\rho(p)$. $\rho(p)$ est le ratio entre le nombre d'arêtes $L(p)$ et le nombre de nœuds ($|\Gamma(p)|$) dans le 1-voisinage $\Gamma(p)$ de p. Les nœuds de $\Gamma(p)$ sont les nœuds gris foncé ($\Gamma(p) = \{a, b, c, d, e, f\}$). $L(p)$ représente alors le nombre de liens entre p et ces nœuds (liens en pointillés) et le nombre de liens entre ces nœuds (liens tiret). Ainsi, $L(p) = 4 + 6 = 10$ et $\delta(p) = 6$ d'où $\rho(p) = 10/6 = 5/3$. On remarquera que pour calculer $\rho_k(p)$, p doit connaître $\Gamma_{k+1}(p)$ afin de connaître les liens existant entre ses k-voisins.

3.3 La formation des clusters

Chaque nœud u surveille son voisinage et juge ainsi de sa mobilité relative. Si cette dernière n'est pas trop importante, alors u participe à l'algorithme de *clustering*, sinon, il reste un nœud indépendant. L'idée est qu'un nœud trop mobile ne pourra pas instancier de communications avec les autres entités du réseau. Il n'a donc pas besoin d'appartenir à un cluster puisqu'il ne pourrait pas en tirer avantage. De même, si un

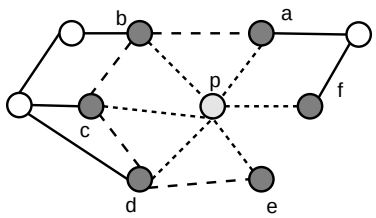


FIG. 3.1 – Illustration de la métrique de densité.

nœud trop mobile est pris en compte dans la construction des clusters, il risque de casser la structure inutilement de par les nombreuses cassures de liens induites par sa forte mobilité et obligera le réseau à reconstruire les clusters. Cette valeur de mobilité relative d'un nœud u peut être calculée en vérifiant la constance du voisinage de u, en considérant par exemple le nombre de nœuds restant dans le voisinage de u pendant un certain temps.

Périodiquement, chaque nœud suffisamment stable calcule sa densité et la diffuse localement à son 1-voisinage. Chacun est alors en mesure de comparer sa propre densité à celle de ses voisins "suffisamment stables". À partir de là, un nœud décide soit de s'élire comme cluster-head (s'il possède la plus forte densité), soit de choisir comme père son voisin de plus forte densité. En cas d'égalité, afin de privilégier la stabilité de la structure, le nœud choisi sera celui déjà élu au tour précédent s'il est en course, sinon celui de plus petit identifiant. De cette façon, deux voisins ne peuvent pas être tous deux cluster-heads. Cette méthode d'élection construit implicitement une forêt couvrante orientée.

Si un nœud u choisit le nœud w, on dit que w est le père de u (noté $\mathcal{P}(u) = w$) dans l'arbre de *clustering* et que u est un fils de w (noté $u \in \mathcal{C}h(w)$). Si aucun nœud n'a élu le nœud u comme père ($\mathcal{C}h(u) = \emptyset$), u est une feuille d'un des arbres, sinon, u est qualifié de nœud interne. Le père d'un nœud peut s'être choisi comme père un autre nœud de son voisinage et ainsi de suite. Un arbre s'étend automatiquement, sans contrainte sur sa hauteur, jusqu'à atteindre les frontières d'un autre arbre. Tous les nœuds appartenant au même arbre appartiennent alors au même cluster. Afin d'apporter une stabilité plus importante, un chef de cluster ne doit pas être trop excentré dans son propre cluster. En effet, si un chef de cluster se trouve à la frontière de son cluster et qu'il bouge, il a plus de chance d'entrer en compétition avec un autre chef et ainsi de casser les deux clusters. C'est pourquoi, nous ajoutons une règle supplémentaire qui indique que tout nœud voisin d'un cluster-head doit s'attacher à ce cluster-head. Si un nœud est voisin de plusieurs cluster-heads, une fusion est instanciée entre ces clusters et le cluster résultant a pour chef le cluster-head en compétition de plus forte densité. De cette façon, deux cluster-heads sont distants de 3 sauts minimum. Supposons un nœud u voisin d'un chef de cluster $\mathcal{H}$ ($\mathcal{H} \in \Gamma_1(u)$) mais qui ne l'a pas choisi comme père ($\mathcal{P}(u) \neq \mathcal{H}$), alors, deux cas sont possibles :
– soit le père de u est également cluster-head ; dans ce cas les deux clusters fusionnent.

Le cluster-head final est le nœud le plus fort parmi ceux en compétition : $\mathcal{P}(u)$. (Puisque u pouvait choisir entre $\mathcal{H}$ et $\mathcal{P}(u)$ et a choisi $\mathcal{P}(u)$). $\mathcal{H}$ n'est plus cluster-head, il choisit u comme son père ;

– soit le père de u s'est attaché à un autre de ses voisins et n'est pas cluster-head ; cela signifie que u se situe à au moins deux sauts de son chef. Il change alors de père et choisit $\mathcal{H}$.

Étant donné un nœud $v \in V$, pour tout nœud $u \in \Gamma_1(v)$, on définit $Age(u)$ comme le nombre de périodes successives où un nœud u a choisi v comme père. On définit également $\prec$ comme un indicateur d'ordre binaire tel que, pour $(u, v) \in V^2$, $u \prec v$ si et seulement si $\{\rho_k(u) < \rho_k(v)\}$ ou $\{\rho_k(u) = \rho_k(v) \wedge Age(u) < Age(v)\}$ ou $\{\rho_k(u) = \rho_k(v) \wedge Age(u) = Age(v) \wedge$ *Id(v) < Id(u)*$\}$.

L'algorithme s'auto-stabilise quand chaque nœud connaît l'identité de son cluster-head.

Algorithm 1 Formation des clusters

Pour tout nœud $u \in V$
▷ *Initialisation des variables.*
$\mathcal{H}(u) = \mathcal{P}(u) = -1$
$\forall v \in \Gamma_1(u), Age(v) = 0$
while ($\mathcal{H}(u) = -1$) ou ($\mathcal{H}(u) \neq \mathcal{H}_{old}(u)$))
▷ *Boucle jusqu'à stabilisation*
$\mathcal{H}_{old}(u) = \mathcal{H}(u)$
Scrutation du voisinage
Calcul de la valeur de mobilité
if ($Mobilite < Seuil_{Mobilite}$)
Récupère $\Gamma_{k+1}(u)$
Calcule $\rho_k(u)$
Diffuse localement $\rho_k(u)$ à ses 1-voisins.
▷ *Cette diffusion locale peut s'effectuer par exemple en ajoutant la valeur de $\rho_k(u)$ dans un paquet* HELLO.
▷ *À ce moment, le nœud u connaît la k-densité de tous ses voisins et peut choisir son père.*
if ($\forall v \in \Gamma_1(u),\ v \prec u$) **then** $\mathcal{H}(u) = u$ ▷ *u devient cluster-head.*
else
▷ $\exists w \in \Gamma_1(u)$ *t.q.* $\forall v \in \{u\} \cup \Gamma_1(u),\ v \prec w$
$\mathcal{P}(u) = w$
$\mathcal{H}(u) = \mathcal{H}(w)$
▷ *Soit $\mathcal{P}(w) = \mathcal{H}(w) = w$ donc u est directement lié à son chef de cluster, soit w a choisi un autre nœud x comme père ($\exists\ x \in \Gamma_1(w) \mid \mathcal{P}(w) = x$) et récursivement $\mathcal{H}(u) = \mathcal{H}(w) = \mathcal{H}(x)$.*
end
if (($\mathcal{H}(u) = u$) et ($\exists v \in \Gamma_1(u) \mid \mathcal{P}(v) \neq u$)) **then**
▷ *u est cluster-head, mais tous ses voisins ne l'ont pas choisi comme père.*
if ($\mathcal{P}(v) = \mathcal{H}(v)$) **then**
▷ *Au moins deux cluster-heads ($\mathcal{H}(u)$ et $\mathcal{H}(v)$) ont un voisin commun v. Si $\mathcal{P}(v) = \mathcal{H}(v)$ alors $u \prec \mathcal{H}(v)$. u s'écrase et choisit v comme père (les clusters $\mathcal{C}(u)$ et $\mathcal{C}(v)$ fusionnent).*
$\mathcal{P}(u) = v$ et $\mathcal{H}(u) = \mathcal{H}(v)$
Age(v)++ et $\forall w \in \Gamma_1(u),\ Age(w) = 0$.
end

end
if ($\exists v \in \Gamma_1(u)$ t.q. $\{(\mathcal{H}(v) = v)$ et $(\mathcal{H}(\mathcal{P}(u)) \neq \mathcal{H}((u))\}$)
▷ u n'est pas chef et est à plus de 2 sauts de son chef (son père n'est pas chef) alors qu'il compte un chef parmi ses voisins. Il change de père.
$\mathcal{P}(u) = v$ et $\mathcal{H}(u) = v$
$Age(v)$++ et $\forall w \in \Gamma_1(u)\ Age(w) = 0$.
end
Diffuse localement $\mathcal{P}(u)$ et $\mathcal{H}(u)$
end

Exemple

Afin d'illustrer cette heuristique, exécutons l'Algorithme 1 sur le graphe de la figure 3.2 en considérant la 1-densité. Dans le 1-voisinage du nœud a, on a deux voisins ($\Gamma_1(a) = \{d, i\}$) et deux liens ($\{(a, d), (a, i)\}$), d'où $\rho(a) = 1$; le voisinage du nœud b compte 4 voisins ($\Gamma_1(b) = \{c, d, h, i\}$) et cinq liens ($\{(b, c), (b, d), (b, h), (b, i), (h, i)\}$), d'où $\rho(b) = \frac{5}{4}$. La table 3.1 montre les valeurs finales des densités des nœuds.

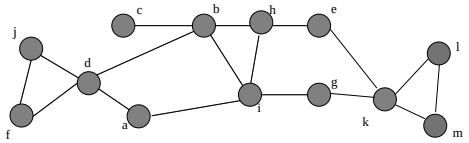
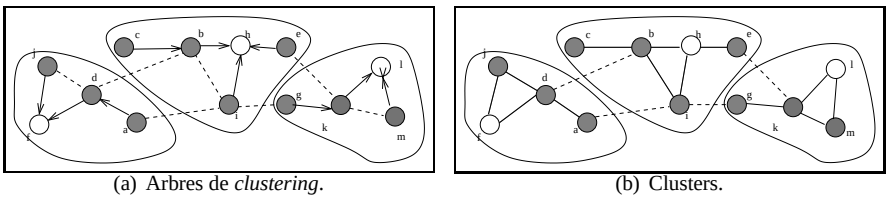


FIG. 3.2 – Exemple.

Nœuds	a	b	c	d	e	f	g	h	i	j	k	l	m
Degré	2	4	1	4	2	2	2	3	4	2	4	2	2
Nb Liens	2	5	1	5	2	3	2	4	5	3	5	3	3
Densité	1	1.25	1	1.25	1	1.5	1	1.33	1.25	1.5	1.25	1.5	1.5

TAB. 3.1 – Densité des nœuds du graphe de la figure 3.2.

Dans cet exemple, le nœud c élit son voisin b ($\mathcal{P}(c) = b$) dont la densité est la plus forte dans $\Gamma_1(c) \cup \{c\}$ ($\forall v \in \Gamma_1(c) \cup \{c\}, v \prec b$). Le nœud de plus forte densité dans le voisinage de b est h, d'où $\mathcal{P}(b) = h$. Comme h a la plus forte densité dans son voisinage, il devient son propre père et donc cluster-head : $\mathcal{H}(h) = h$. Le nœud c choisit b qui choisit h et tous trois appartiennent au même cluster de cluster-head h et donc : $\mathcal{H}(c) = \mathcal{H}(b) = \mathcal{H}(h) = h$. $\rho(j) = \rho(f)$: ni j ni f n'étaient choisis auparavant, c'est donc le plus petit identifiant qui tranche. Supposons $j \prec f$, alors $\mathcal{P}(j) = f$ et $\mathcal{P}(f) = f$ d'où $\mathcal{H}(f) = \mathcal{H}(j) = f$. Aucun nœud n'ayant choisi a, j, c, e, i, g et m comme père, ils deviennent des feuilles. Finalement, nous obtenons une forêt couvrante du réseau, composée de trois arbres de racines h, l et f (figure 3.3(a)), qui donnent naissance à trois clusters (figure 3.3(b)).



(a) Arbres de *clustering*. (b) Clusters.

FIG. 3.3 – Arbres (a) et clusters (b) construits avec l'Algorithme 1 sur le graphe de la figure 3.2 (Les cluster-heads/racines apparaissent en blanc).

3.4 Maintenance de la structure

La maintenance de cette structure construite à partir de l'heuristique de la k-densité est simple, étant donné que chaque nœud n'a besoin que de son $k+1$-voisinage pour la construire. En effet, d'après la taxonomie établie par [81], un algorithme peut être qualifié de local si chaque nœud n'a besoin que de la connaissance de son 1 et 2 voisinage pour l'exécuter, ou de quasi-local, si les nœuds nécessitent une information dans un voisinage borné. Cela implique une maintenance rapide. Chaque nœud calcule périodiquement ses valeurs de mobilité et de densité. S'il est suffisamment stable, il compare sa densité à celle de ses voisins et choisit périodiquement son père. Un nœud non stable à l'origine et dont le voisinage se stabilise peut ainsi s'attacher à la structure sans la détruire.

3.5 Analyse de la métrique

Dans cette section, nous nous sommes intéressés aux différentes caractéristiques de notre métrique de densité.

Dans un premier temps, nous avons calculé sa valeur théorique moyenne à l'aide de la géométrie stochastique et des calculs de Palm. Puis, nous avons comparé les différentes k-densités. Nous verrons ainsi que la 1-densité est non seulement la densité la moins coûteuse mais également la plus stable face à la mobilité des nœuds et que les cluster-heads sélectionnés agissent comme des bassins d'attraction. Nous terminerons cette partie par une analyse de la répartition des valeurs de densité parmi les nœuds du réseau.

3.5.1 Recherche de la meilleure k-densité

Nous nous sommes interrogés sur les différentes k-densités : laquelle est la plus adéquate ? En effet, nous avons vu que pour calculer une k-densité, chaque nœud doit connaître son $k+1$ voisinage. Ainsi, plus k augmente, plus la k-densité est coûteuse en

messages, utilisation de bande passante et latence. C'est pourquoi, nous avons comparé par simulation les structures formées par la 1-densité et la 2-densité.

Comme le montre la table 3.2, la 2-densité construit moins de clusters que la 1-densité. Un nœud est plus excentré dans son cluster avec $k = 2$. Néanmoins, ces caractéristiques très similaires ne nous permettent pas de trancher entre les différentes densités. C'est pourquoi, nous avons également comparé le comportement des structures obtenues avec les densités 1 et 2 face à la mobilité des nœuds. En effet, la densité la plus intéressante sera celle qui offre la meilleure stabilité, *c.à.d* qui reconstruit moins souvent les clusters lorsque les nœuds se déplacent, limitant ainsi les échanges de messages de contrôle et de mise à jour des tables de routage. Un nœud peut quitter son cluster et migrer dans un autre sans que cela ne casse la structure de clusters.

	750 nœuds		3000 nœuds		5000 nœuds	
k-densité	1	2	1	2	1	2
Nb clusters	4.67	3.01	4.23	2.53	4.42	2.43
$D(\mathcal{C})$	7.1	9.72	9.25	11.67	9.4	12.15
$\tilde{e}(u/\mathcal{C})$	4.86	6.45	6.21	7.03	6.02	8.42

TAB. 3.2 – Comparaison des k-densités.

Pour cela, nous avons simulé un réseau où les nœuds peuvent choisir aléatoirement de bouger à différentes vitesses allant de 0 à $1.6m/s$ (piétons) dans des directions aléatoires (Modèle de mobilité Random Way Point) pendant $500s$. La table 3.3 donne le nombre moyen de reconstructions de clusters durant la simulation.

	Moy	Min	Max
1-densité	7.5	2	13
2-densité	9.4	4	14

TAB. 3.3 – Nombre de clusters re-contruits après mobilité des nœuds.

La 2-densité reconstruit plus souvent la structure que la 1-densité. La 1-densité s'avère donc la densité la plus robuste et la moins coûteuse puisqu'elle ne nécessite la connaissance que du 2-voisinage d'un nœud, tout comme dans OLSR. De plus, utiliser une k-densité avec $k > 2$ serait trop coûteux et impliquerait une maintenance moins efficace. C'est pourquoi, par la suite, on ne considérera que la 1-densité.

3.5.2 Densité moyenne

Nous analysons ici la 1-densité moyenne $\tilde{\rho}(u)$ d'un nœud u. On considère un réseau sans fil multi-sauts où les nœuds sont distribués suivant un processus de Points de Poisson d'intensité constante λ. Nous calculons alors la 1-densité moyenne en considérant une distribution de Palm. Dans une telle distribution, un nœud 0 est artificiellement

ajouté à la distribution poissonienne. Ce nœud 0, placé à l'origine du plan, sert de base d'observation pour les calculs. Sous la probabilité de Palm, ce nœud existe presque sûrement. Puisque le processus est stationnaire, la densité moyenne de 0 est valide pour tout autre point du processus.

Soit $\rho(0)$ la densité moyenne du nœud 0. Φ désigne le processus ponctuel : $\Phi(S)$ représente le nombre de points du processus se trouvant sur une surface S donnée. Soit $B(u,R)$ la boule de centre u et de rayon R et $B_u^{'}$ la boule centrée en u de rayon R, privée du singleton $\{u\}$: $B_u^{'} = B(u,R)\backslash\{u\}$. $\mathbb{E}^o$ et $\mathbb{P}^o$ désignent respectivement l'espérance et la probabilité sous la distribution de Palm.

Nous cherchons donc à calculer la densité moyenne $\tilde{\rho}(u) = \mathbb{E}^o[\rho(0)]$ d'un nœud quelconque u.

Lemme 1 *La* 1*-densité moyenne de tout nœud* u *s'écrit :*

$$\tilde{\rho}(u) = \mathbb{E}^o[\rho(0)] = 1 + \frac{1}{2}\left(\pi - \frac{3\sqrt{3}}{4}\right)\left(\lambda R^2 - \frac{1 - exp\{-\lambda\pi R^2\}}{\pi}\right)$$

La preuve de ce lemme est donnée en annexes 3.11.1. L'idée est de compter le nombre de liens dans le voisinage d'un nœud. Un lien existe entre deux nœuds s'ils sont à une distance inférieure ou égale à R. Si v est un voisin de u, alors, il existe autant de liens entre v et un autre voisin de u que de voisins communs à u et v (nœuds à distance inférieure à R à la fois de u et de v). Le nombre de voisins communs à u et v correspond au nombre de points se trouvant à l'intersection des zones de transmission de u et v. Cette surface est la zone représentée en bleu sur la figure 3.4. Par la suite, nous dénoterons par $A(r)$ cette surface. Le calcul de $\tilde{\rho}$ consiste à sommer le nombre de points se trouvant en moyenne dans cette zone pour chacun des voisins v de u en fonction de la distance euclidienne r de u à v. Les liens étant ainsi comptés deux fois, la somme finale est divisée par deux.

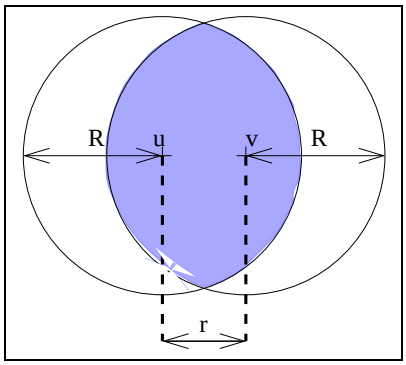


FIG. 3.4 – Intersection du voisinage de deux nœuds voisins u et v. Les nœuds se trouvant dans la zone bleue sont des voisins communs à u et v.

Afin de vérifier la validité de nos résultats analytiques, nous avons comparé les valeurs de densité moyenne obtenues par analyses et par simulation. Le modèle de simulation utilisé est celui décrit dans le Chapitre 1.1. La table 3.4 donne les résultats pour une valeur de $R = 0.1$ et différentes valeurs d'intensité λ du processus de Poisson. Notons que la théorie et la simulation s'accordent parfaitement.

	500 nœuds		600 nœuds		700 nœuds	
	Théorie	Simulation	Théorie	Simulation	Théorie	Simulation
$\tilde{\bar{\delta}}$	15.7	15.3	18.8	18.3	22.0	21.2
$\tilde{\rho}$	4.7	5.0	5.6	5.9	6.5	6.8
	800 nœuds		900 nœuds		1000 nœuds	
	Théorie	Simulation	Théorie	Simulation	Théorie	Simulation
$\tilde{\bar{\delta}}$	25.1	25.0	28.3	27.9	31.4	31.0
$\tilde{\rho}$	7.5	7.1	8.4	8.6	9.3	9.4

TAB. 3.4 – Degré et densité moyens des nœuds.

3.5.3 Répartition des valeurs de densité

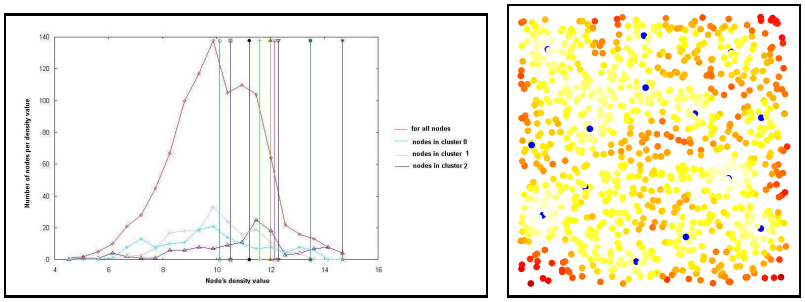
La figure 3.5 montre comment les valeurs de densité sont réparties. La figure 3.5(a) donne le nombre de nœuds ayant une valeur de densité donnée. Les barres verticales indiquent les valeurs prises par les cluster-heads. La figure 3.5(b) donne un exemple de distribution des densités dans le plan. Les cluster-heads apparaissent en bleu. Plus la couleur des nœuds est jaune, plus leur densité est forte. Nous pouvons constater que dans chaque cluster, les densités les plus fortes se situent autour des chefs de cluster. Plus un nœud est loin d'un cluster-head, plus sa densité est faible. Les cluster-heads forment des sortes de bassin d'attraction, ce qui apporte une stabilité à la structure.

3.6 Analyse de la structure

Afin de pouvoir au mieux utiliser la structure de clusters formée par notre heuristique, nous en avons étudié certaines caractéristiques par simulation et quand nous le pouvions, par analyse théorique en utilisant la géométrie stochastique.

3.6.1 Analyse théorique du nombre de clusters

Dans cette section, nous avons cherché à calculer analytiquement le nombre de clusters (ou de cluster-heads) produit par l'algorithme de *clustering* (algorithme 1). Comme pour le calcul de la densité moyenne (section 3.5.2), nous utilisons la géométrie stochastique suivant le modèle défini dans la section 1.1.



(a) Histogramme. Nombre de nœuds du réseau ayant une densité donnée.

(b) Plus la couleur est rouge, plus la densité est grande. Cluster-heads en bleu.

FIG. 3.5 – Distribution des valeurs de densité parmi les nœuds. Les cluster-heads apparaissent en bleu. Plus la couleur des nœuds est jaune, plus leur densité est forte.

Nous utilisons les mêmes notations et la même modélisation que précédemment, à savoir que l'on considère un réseau sans fil multi-sauts où les nœuds sont distribués suivant un processus ponctuel de Poisson Φ d'intensité constante λ.

On cherche dans un premier temps à calculer le nombre de clusters (ou cluster-heads) dans un espace C.

Lemme 2 *Le nombre moyen de cluster-heads appartenant à un domaine C est :*

$$\mathbb{E}\left[\text{Nombre de cluster-heads dans } C\right] = \lambda\nu(C)\mathbb{P}^o_\Phi\left(0 \text{ est chef }\right)$$

où $\nu(C)$ représente la mesure de Lebesgue[1] *dans $\mathbb{R}^2$.*

Pour déterminer le nombre moyen de cluster-heads, il nous faut donc dans un premier temps calculer $\mathbb{P}^o_\Phi\left(0 \text{ est chef }\right)$, probabilité qu'un nœud soit chef. Les nœuds étant distribués uniformément et indépendamment, cette probabilité est la même pour tous les nœuds. Cela revient à calculer la probabilité qu'un nœud ait la plus forte densité dans son voisinage.

Lemme 3 *La probabilité qu'un point 0 soit chef sous la probabilité de Palm est :*

$$\mathbb{P}^o_\Phi\left(0 \text{ est chef}\right) \quad = \quad \mathbb{P}^o_\Phi\left(\rho(0) > \max_{k=1,..,\Phi(B_0)} \rho(Y_k)\right)$$

Nous avons cherché à calculer cette quantité mais n'avons pu obtenir qu'une borne supérieure.

[1] La mesure de Lebesgue sur $\mathbb{R}$ d'un intervalle coïncide avec sa longueur, la mesure de Lebesgue d'une région de l'espace sur $\mathbb{R}^2$ coïncide avec la surface de cet espace. Pour une définition plus formelle, se référer à [76], chapitre 1.

Conjecture 1 *Une borne supérieure pour la probabilité qu'un nœud soit chef est :*

$$\mathbb{P}^o_\Phi\left(\rho(0) > \max_{k=1,..,\Phi(B_0)} \rho(Y_k)\right) \leq \left(1 + \sum_{n=1}^{+\infty} \frac{1}{n} \frac{\left(\lambda\pi R^2\right)^n}{n!}\right) \exp\left\{-\lambda\pi R^2\right\}$$

Les détails du calcul sont donnés en annexes 3.11.2. L'idée est dans un premier temps de conditionner la probabilité que 0 soit chef par le fait qu'il ait ou non des voisins. Si 0 n'a pas de voisin, 0 est chef avec la probabilité 1. Dans le cas contraire, on majore cette probabilité par la probabilité qu'un voisin v de 0 ait la plus forte densité parmi $\Gamma(0)$, les voisins de 0, probabilité que l'on peut calculer sous Palm.

La figure 3.6 représente la borne supérieure du nombre de clusters pour différentes valeurs de R et de λ (avec R diminuant de bas en haut). On peut voir que lorsque que l'intensité des nœuds augmente, la borne supérieure du nombre de clusters tend asymptotiquement vers une constante, ce qui permet à notre structure de supporter le passage à l'échelle du réseau.

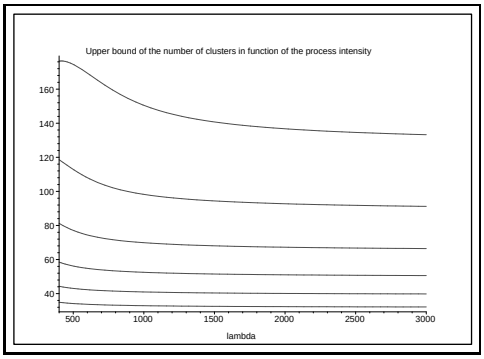


FIG. 3.6 – Borne supérieure du nombre de clusters en fonction de λ (en abscisse) et de R (différentes courbes : de bas en haut $R = 0.1, 0.09, 0.08, 0.07, 0.06, 0.05$ m).

3.6.2 Caractéristiques des clusters

Les résultats donnés dans cette section ont été obtenus par simulation, en utilisant le modèle décrit dans la section 1.1. La table 3.5 résume les caractéristiques principales des clusters pour $R = 0.1$ et différentes valeurs de λ.

On remarquera que malgré l'augmentation de l'intensité des nœuds, l'excentricité moyenne $\tilde{e}(u/\mathcal{C})$ d'un nœud dans son cluster et la hauteur moyenne des arbres de *clustering* restent constante, du fait du nombre constant de clusters. Ceci va s'avérer être un atout lors de l'utilisation de notre structure pour effectuer une diffusion (cf. chapitre 4) ou pour router sur cette structure (cf. chapitre 5). Nous pouvons également noter qu'une grande partie des nœuds sont des feuilles dans l'arbre de *clustering* (environ 75%). Comme nous le verrons dans le chapitre 4, cette propriété va elle aussi s'avérer fort utile lors d'une diffusion d'un message sur une telle structure.

	500 nœuds	600 nœuds	700 nœuds
# clusters/arbres	11.76	11.51	11.45
$\tilde{e}(u/\mathcal{C})$	3.70	3.75	3.84
$\tilde{e}(\mathcal{H}(u)/\mathcal{C}(u))$	3.01	3.09	3.37
Hauteur des arbres	3.27	3.34	3.33
% feuilles	73,48%	74,96%	76,14%
Degré dans l'arbre des nœuds non feuilles	3.82	3.99	4.19
Voronoï : distance euclidienne	84.17%	84.52%	84.00%
Voronoï : nombre de sauts	85.43%	84.55%	84.15%
	800 nœuds	900 nœuds	1000 nœuds
# clusters/arbres	11.32	11.02	10.80
$\tilde{e}(u/\mathcal{C})$	3.84	3.84	3.84
$\tilde{e}(\mathcal{H}(u)/\mathcal{C}(u))$	3.17	3.19	3.23
Hauteur des arbres	3.34	3.43	3.51
% feuilles	76,81%	77,71%	78,23%
Degré dans l'arbre des nœuds non feuilles	4.36	4.51	4.62
Voronoï : distance euclidienne	83.97%	83.82%	83.70%
Voronoï : nombre de sauts	83.80%	83.75%	83.34%

TAB. 3.5 – Caractéristiques des clusters.

Forme des clusters.

Comme le montre la figure 3.7, les clusters ressemblent à un diagramme de Voronoï construit autour des chefs de clusters. Si S est un ensemble de n sites de l'espace euclidien, pour chaque site p de S, la cellule de Voronoï $V(p)$ de p est l'ensemble des points de l'espace qui sont géographiquement plus proches de p que de tous les autres sites de S. Le diagramme de Voronoï de $V(S)$ est la décomposition de l'espace en cellules de Voronoï des sites de l'espace.

Ainsi, cela signifierait qu'étant donnés les cluster-heads, un nœud s'est attaché à celui le proche de lui en distance euclidienne. Afin d'évaluer cette caractéristique, nous avons mené des simulations pour connaître le pourcentage de nœuds se situant dans la cellule de Voronoï de leur chef et étant donc plus près de lui que de tout autre chef en distance euclidienne. De plus, comme dans un réseau sans fil, on ne considère pas la distance euclidienne mais la distance en nombre de sauts, nous avons également regardé quelle proportion de nœuds étaient plus proches en nombre de sauts de leur propre chef plutôt que de tout autre. Les résultats numériques sont donnés dans la table 3.5. La figure 3.8 donne pour une topologie de clusters (figure 3.8(a)) la proportion des nœuds se situant dans la "bonne" cellule de Voronoï en distance euclidienne (figure 3.8(b)) et en nombre de sauts (figure 3.8(c)). On remarquera que plus de 80% sont plus proches de leur cluster-head que d'un autre aussi bien en distance euclidienne qu'en nombre de sauts. Ceci présente un avantage également pour la diffusion d'un message dans un cluster, comme nous le verrons plus tard dans le chapitre 4.

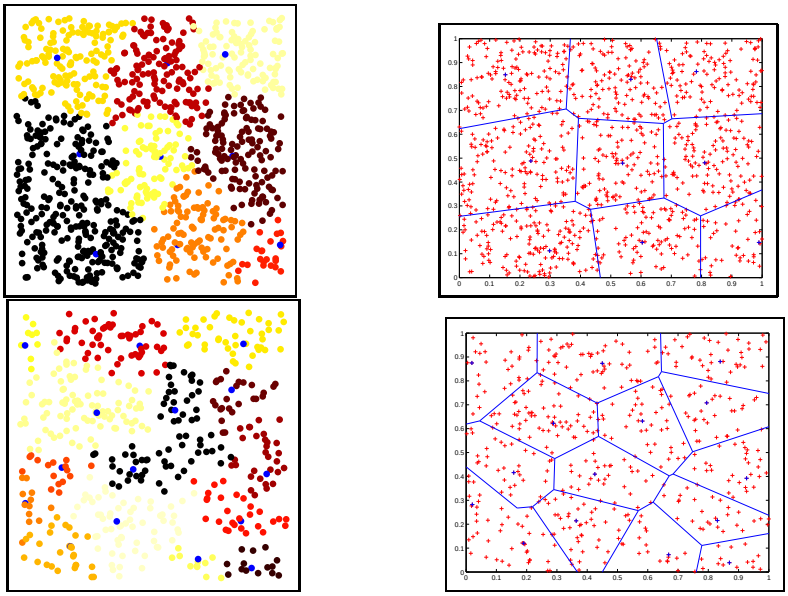
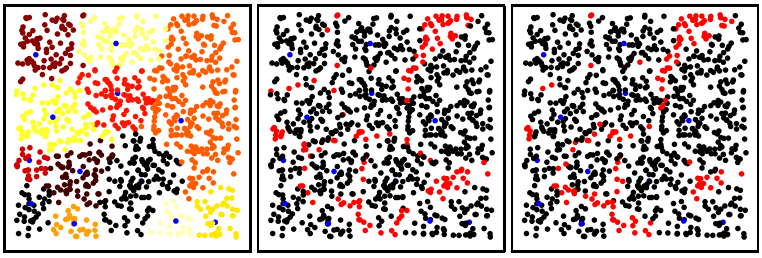


FIG. 3.7 – Structure de clusters (schémas de gauche) et diagramme de Voronoï correspondants (schémas de droite) pour $\lambda = 1000$ et $\lambda = 500$.



(a) Topologie (b) Voronoï euclidien (c) Voronoï sauts

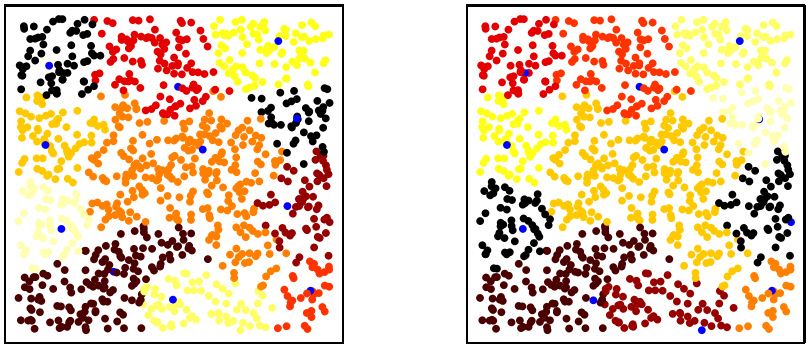
FIG. 3.8 – Pour une structure de clusters (a), les nœuds dans la ”bonne” cellule de Voronoï en distance euclidienne (b) ou en nombre de sauts (c) apparaissent en noir.

3.7 Comparaison à d'autres heuristiques

Dans le but d'évaluer notre heuristique et d'être en mesure de la situer parmi les heuristiques existantes, nous la comparons à d'autres heuristiques de la littérature : DDR [59] et Max-Min d-cluster [4]. Ces heuristiques, décrites dans le chapitre 2 construisent des clusters dont le rayon est supérieur à 1 saut comme notre heuristique. Max-Min d-cluster utilise l'identifiant des nœuds mais cherche à ne pas toujours favoriser l'identifiant le plus fort et ainsi éviter que les plus grands identifiants soient toujours cluster-heads. DDR, quant à lui, est très semblable à notre algorithme mais se base sur le degré des nœuds.

3.7.1 Comparaison avec DDR

L'heuristique de DDR [59] construit des clusters de façon assez similaire à la nôtre mais utilise le degré des nœuds comme métrique au lieu de la densité. Tout comme dans notre cas, le rayon des clusters n'est pas fixé à l'avance et s'adapte automatiquement à la topologie sous-jacente. Le degré est moins coûteux que la densité puisqu'il nécessite la connaissance du 1-voisinage seulement. C'est pourquoi, nous avons voulu comparer les structures de clusters obtenues pour chacune des heuristiques.



(a) Clusters DDR (b) Clusters Densité

FIG. 3.9 – Exemple de structure obtenue pour $\lambda = 1000$ et $R = 0.1$ avec DDR (a) et avec la 1-densité (b).

	Nb clusters	Nb nœuds par cluster	$\bar{D}(\mathcal{C})$	$\tilde{e}(u/\mathcal{C})$
DDR	10.0	100	5.8	4.2
densité	11.1	90.9	5.0	3.8

TAB. 3.6 – Comparaison des clusters de DDR et de ceux obtenus par notre heuristique.

Les résultats sont donnés par la figure 3.9 et la table 3.6. On remarquera que les structures sont très semblables. Nous avons alors comparé la robustesse des structures face à la mobilité des nœuds. Afin de permettre le passage à l'échelle de la structure et de limiter les échanges entre les nœuds, les chefs de cluster doivent rester chefs aussi longtemps que possible tout en maintenant des clusters adaptés à la topologie sous-jacente. Les auteurs de [66] donnent la définition suivante de la robustesse : *"one measure of robustness of the topology is given by the maximum number of nodes that need to change their topology information as a result of a movement of a node"*[2]. C'est pourquoi, nous avons mené des simulations en appliquant une mobilité sur les nœuds et relevé la proportion de chefs étant ré-élus. Plus ce ratio est grand, moins importants sont les changements des informations des tables de routage des nœuds.

Dans nos simulations, chaque nœud peut bouger aléatoirement dans une direction aléatoire à une vitesse aléatoire allant de 0 à $10m/s$ (modèle voiture) et de 0 à $1.6m/s$ (modèle piéton) durant 15 minutes (Modèle Random Way Point). La table 3.7 donne le pourcentage de cluster-heads ré-élus toutes les 2 secondes par chacune des heuristiques. Les résultats montrent qu'en moyenne, notre heuristique reconstruit moins souvent les clusters que DDR. Elle s'avère donc être plus robuste.

	500 nœuds		600 nœuds		800 nœuds		1000 nœuds	
	ρ	DDR	ρ	DDR	ρ	DDR	ρ	DDR
$1.6m/s$	68.7%	65%	67.2%	63.5%	64.5%	62.4%	62.2%	56.8%
$10m/s$	30.1%	27.5%	27%	25.3%	26.2%	23.1%	24.8%	20.35%

TAB. 3.7 – % de cluster-heads ré-élus.

De par la mobilité des nœuds, un nouveau nœud peut apparaître dans le voisinage d'un autre. De façon à comprendre pourquoi la métrique de densité est plus stable que le degré utilisé par DDR, nous avons analysé comment le voisinage d'un nœud est perturbé par l'apparition de ce nouveau nœud. La structure de clusters se re-construit si le nœud qui avait le plus fort degré ou la plus forte densité dans son voisinage se trouve un voisin dont le degré ou la densité est devenu(e) plus fort(e) que le/la sien(ne), donc si l'ordre des degrés ou densités entre les nœuds a changé.

Nous considérons un processus ponctuel de Poisson d'intensité λ, distribué dans une boule de rayon $2R$ centrée en un point 0 : $B(0, 2R)$. Nous considérons alors le degré et la densité de ce point 0 ainsi que ceux d'un de ses voisins y choisi arbitrairement. Nous ajoutons alors le nœud mobile u dans le voisinage de 0. u est un nœud uniformément distribué dans $B(0, R)$. Nous pouvons alors calculer la probabilité que l'ordre des degrés de 0 et y change à cause de la présence de u. Cela ne peut se faire que si u n'est pas voisin de y et si $\delta(0) = \delta(y)$ ou $\delta(0) = \delta(y) - 1$. En effet, si u est voisin à la fois de 0 et de y, l'ordre ne changera pas. Si u n'est voisin que de 0 et si $\delta(0) > \delta(y)$, seul le degré de 0 augmente, ce qui ne change pas l'ordre. De même si $\delta(0) < \delta(y) - 1$, même si le degré de 0 augmente de 1, il reste inférieur à $\delta(y)$.

[2]Une mesure de la robustesse de la topologie est donnée par le nombre maximum de nœuds qui doivent changer leur information sur la topologie pour le mouvement d'un nœud.

Soit C une variable désignant la région du plan $B(0,R)\backslash B(y,R)$. C correspond à la zone de voisinage du nœud 0 dans laquelle les nœuds ne sont pas voisins de y (Voisinage de 0 non coloré sur la figure 3.4 où on prend $u=0$ et $v=y$). $\nu(C)$ est la mesure de Lebesgue de $C=B(0,R)\backslash B(y,R)$.

La probabilité P_1 que u ne soit voisin que du nœud 0 revient à la probabilité que u se trouve dans la zone de C. D'où : $P_1=\frac{\nu(C)}{\pi R^2}$.

La probabilité P_2 que $\delta(0)=\delta(y)$ est la probabilité que 0 et y aient autant de voisins non communs avant l'arrivée de u. D'où :

$$\begin{aligned} P_2 &= \sum_{k=0}^{\infty}\mathbb{P}(\Phi(B(0,R)\backslash B(y,R))=k)\times\mathbb{P}(B(y,R)\backslash B(0,R)=k) \\ &= \sum_{k=0}^{\infty}\frac{\lambda^{2k}\nu(C)^{2k}}{k!k!}\exp -2\lambda\nu(C) \end{aligned}$$

La probabilité P_3 que $\delta(0)=\delta(y)-1$ est la probabilité que y ait un voisin non commun avec 0 en plus de 0. D'où :

$$\begin{aligned} P_3 &= \sum_{k=0}^{\infty}\mathbb{P}(\Phi(B(0,R)\backslash B(y,R))=k)\times\mathbb{P}(\Phi(B(y,R)\backslash B(0,R))=k+1) \\ &= \sum_{k=0}^{\infty}\frac{\lambda^{2k+1}\nu(C)^{2k+1}}{k!(k+1)!}\exp -2\lambda\nu(C) \end{aligned}$$

À partir de là, pour obtenir la probabilité P_p que l'ordre change, il nous faut faire la moyenne sur tous les points y, en multipliant par la probabilité que y existe, *c.à.d.* par la probabilité $\mathbb{P}(\Phi(B(0,R))>0)$ que le nœud 0 ait au moins un voisin. Sachant que y est uniformément distribué dans $B(0,2R)$, la probabilité P_p que l'ordre change peut alors s'écrire :

$$\begin{aligned} P_p &= \mathbb{E}\left[P_1(P_2+P_3)\right]\mathbb{P}(\Phi(B(0,R))>0) \\ &= \mathbb{E}\left[P_1(P_2+P_3)\right]\left(1-\exp\left\{-\lambda\pi R^2\right\}\right) \\ &= \mathbb{E}\left[\frac{\nu(C)}{\pi R^2}\sum_{n=0}^{+\infty}\frac{\left(\lambda\nu(C)\right)^{2n}}{n!n!}(1+\frac{\lambda\nu(C)}{n+1})\right]\left(1-\exp\left\{-\lambda\pi R^2\right\}\right) \end{aligned}$$

Malheureusement, nous n'avons pas été en mesure de donner un résultat analytique donnant la probabilité que l'ordre des densités des nœuds 0 et y soit perturbé par l'arrivée du nœud u. Néanmoins, nous avons obtenu une approximation par simulation. La figure 3.10 donne les probabilités que l'ordre change pour les deux métriques (obtenues par simulation) et la probabilité P_p obtenue analytiquement pour le degré.

Ces résultats correspondent à ceux obtenus précédemment dans la table 3.7. La probabilité que l'ordre des métriques change avec l'apparition d'un nœud mobile dans un voisinage est plus importante pour le degré que pour la densité. Cela tend à prouver que la structure basée sur la densité est plus stable que celle basée sur le degré.

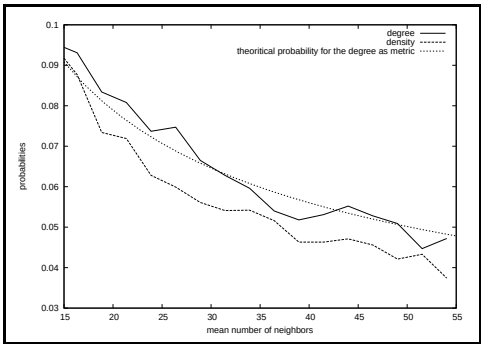


FIG. 3.10 – Probabilité que l'ordre des métriques change entre deux voisins.

3.7.2 Comparaison avec l'heuristique Max-Min d-cluster

L'heuristique Max-Min d-cluster [4] produit également des clusters dont le rayon est supérieur à 1. La structure de Max-Min a obtenu de très bons résultats de stabilité. Elle utilise l'identifiant des nœuds mais tente de contrebalancer le fait qu'un cluster-head élu sur cette métrique garde son rôle quasi-indéfiniment, en élisant non pas le plus grand ou plus petit identifiant mais le nœud possédant le plus petit identifiant parmi les plus grands identifiants des nœuds se trouvant à au plus d sauts de lui. Le paramètre d est le rayon des clusters et doit être fixé *a priori*.

Structure.

	Nb clusters	Nb de nœuds par cluster	$D(\mathcal{C})$	$\tilde{e}(u/\mathcal{C})$
densité	11.1	90.9	5.0	3.8
Max-Min 2-cluster	28.6	34.9	3.6	3.1
Max-Min 3-cluster	13.3	75.2	4.9	3.4
Max-Min 4-cluster	8.2	122.0	6.5	4.9

TAB. 3.8 – Caractéristiques des clusters de la densité et de Max-Min d-cluster.

Dans un premier temps, nous avons simulé Max-Min d-cluster pour plusieurs valeurs du rayon d. Les résultats donnés dans la table 3.8 nous montrent que l'heuristique Max-Min pour $d = 3$ est la plus proche de notre heuristique. Par la suite, c'est celle que nous considérerons.

La figure 3.11 donne le nombre de clusters produits par notre heuristique et par Max-Min 3-cluster pour $\lambda = 1000$ et différentes valeurs de R.

On remarquera que le nombre de clusters obtenu est similaire pour les deux métriques mais Max-Min 3-cluster construit des clusters plus petits lorsque le réseau est peu

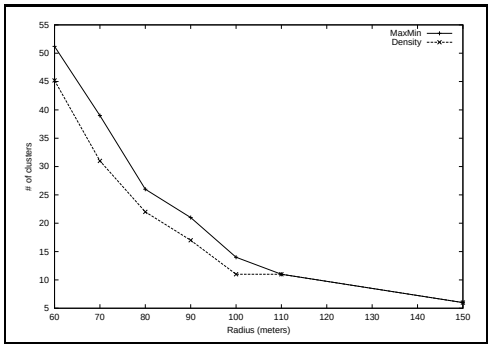
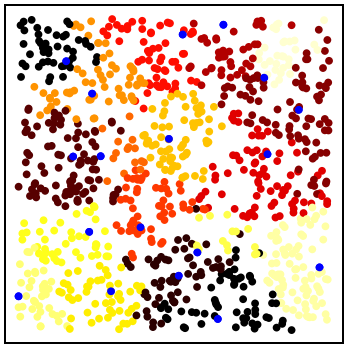


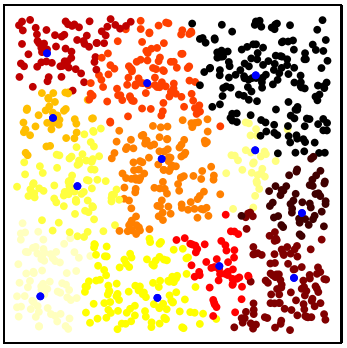
FIG. 3.11 – Nombre de clusters formés pour différentes valeurs de R avec la densité $(- \times -)$ et Max-Min 3-cluster $(- + -)$.

dense. Notre heuristique s'adapte mieux aux réseaux peu denses puisqu'il produit des clusters plus adaptés et en plus petit nombre. De plus, contrairement à l'heuristique de Max-Min, notre algorithme n'autorise pas la formation de clusters à un seul nœud qui sont inutiles.

Les figures 3.12 (a) et (b) donnent un exemple de structure de clusters obtenue par simulation par chacune des heuristiques sur une même distribution des nœuds. Dans les deux cas, les clusters semblent homogènes. Les chefs de clusters sont bien répartis dans l'espace.



(a) Clusters obtenus par Max-Min 3-cluster



(b) Clusters obtenus par notre heuristique

FIG. 3.12 – Exemple d'une structure de clusters pour une topologie à 1000 nœuds de rayon de transmission $R = 0.1$ obtenue avec Max-Min 3-cluster (a) et avec l'heuristique utilisant la densité (b).

Comparaison face à la mobilité des nœuds.

De la même façon que pour DDR, nous avons comparé l'heuristique de Max-Min et la nôtre face à la mobilité des nœuds, en effectuant des simulations où les nœuds bougeaient à différentes vitesses aléatoires. Nous avons pu constater que Max-Min d-cluster ré-élit les mêmes cluster-heads dans plus de 90% des cas. Cela était prévisible étant donné que l'élection considère l'identité des nœuds qui ne change pas lorsque les nœuds bougent. En se basant seulement sur ces résultats, on pourrait donc prétendre que Max-Min est plus stable que notre heuristique. Seulement, l'identifiant des nœuds étant indépendant de la topologie sous-jacente, les clusters nouvellement reformés ne sont pas toujours adaptés à la topologie. Afin d'évaluer cela, nous avons considéré plus en détail l'apparition des nœuds.

Dans les résultats suivants, présentés dans la table 3.9, nous considérons une topologie initiale de 500 nœuds dans laquelle apparaissent progressivement de façon aléatoire, dix vagues de 100 nouveaux nœuds, avec des identifiants aléatoires. Ces résultats donnent le pourcentage de cluster-heads ré-élus et le pourcentage d'augmentation du nombre de clusters dans le réseau.

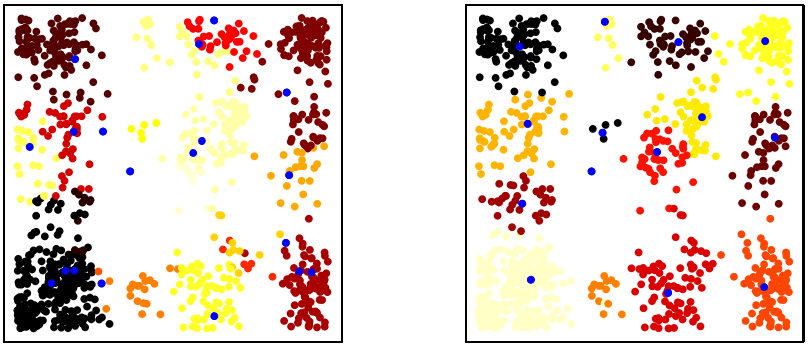
	% cluster-heads ré-élus	Evolution du nombre de clusters
densité	94.3%	$+0\%$
Max-Min 3-cluster	100%	+46%

TAB. 3.9 – Comparaison de Max-Min 3-cluster et de l'heuristique de densité face à l'arrivée des nœuds.

On remarque que même si Max-Min ré-élit toujours les mêmes chefs de cluster, l'heuristique en élit également d'autres. L'heuristique de densité quant à elle incorpore les nouveaux nœuds dans les clusters existants. Le comportement de Max-Min est dû au fait que si un nouveau nœud a un identifiant supérieur au chef déjà en place, il créé son propre cluster en ne modifiant les clusters existants que s'il est dans le voisinage d'un ancien chef. Dans le cas contraire, les anciens clusters restent inchangés. Le nouveau nœud étant souvent le seul nœud de son cluster nouvellement formé.

Comparaison des structures sur des topologies non uniformes.

Enfin, nous avons comparé les structures obtenues par notre heuristique et Max-Min sur des topologies de nœuds non uniformes. Les nœuds sont distribués autour de quelques points choisis aléatoirement qui pourraient représenter des villes. Les figures 3.13 (a) et (b) illustrent une telle topologie. On peut remarquer que notre heuristique génère moins de clusters avec des cluster-heads mieux centrés. Max-Min génère là aussi des clusters à 1 nœud ou des clusters-heads voisins. Par exemple, sur une même topologie de 1000 nœuds, notre heuristique génère 8.7 clusters en moyenne contre 15.25 clusters pour Max-Min.



(a) Clusters obtenus par Max-Min 3-cluster

(b) Clusters obtenus par notre heuristique

FIG. 3.13 – Distribution non uniforme de nœuds : clusters obtenus avec Max-Min 3-cluster (a) et avec la métrique de densité (b).

Complexité.

L'heuristique de Max -Min d-cluster se compose de 3 phases de diffusion de messages à d sauts : une phase récupérant le plus grand identifiant à d sauts, une phase récupérant le plus petit identifiant parmi les plus grands à d sauts et enfin une phase pour diffuser l'identité du chef de cluster. Notre heuristique, quant à elle, est purement locale et ne nécessite qu'une information sur le 2-voisinage obtenue par des messages diffusés uniquement dans le 1-voisinage. Max-Min d-cluster s'avère donc plus coûteux en terme de messages et de latence que notre algorithme.

3.8 Analyse de l'auto-stabilisation

Comme nous avons pu le constater dans le chapitre 2, il existe de nombreux protocoles de *clustering* pour les réseaux sans fil. Cependant, seulement très peu vérifient la robustesse de leur algorithme et, même quand c'est le cas, l'évaluation est menée par simulation et jamais via une analyse théorique. Dans cette section, nous appliquons les principes d'auto-stabilisation à notre algorithme de *clustering*. L'auto-stabilisation est la propriété d'un système à atteindre seul une configuration dans laquelle il a un comportement correct, en partant de n'importe quelle configuration arbitraire. À l'aide d'une approche théorique, nous montrons que, sous certaines hypothèses, l'algorithme est auto-stabilisant localement et que le temps de convergence est faible et borné. Nous validons ensuite cette propriété par simulation.

3.8.1 Pré-requis

Nous présentons dans un premier temps les différentes hypothèses. Nous considérons que l'algorithme se stabilise lorsque chaque nœud connaît l'identité de son cluster-head. Le temps de stabilisation est donc lié à la hauteur des arbres de *clustering*, l'identité du chef devant être transmise jusqu'aux feuilles de l'arbre. Nous suivons les mêmes principes et hypothèses que dans [38] :

Hypothèses. Nous supposons qu'il existe une constante $\tau > 0$ telle que la probabilité qu'un paquet soit transmis sans collision entre deux nœuds voisins est au moins τ. Cela implique que nous supposons que tous les nœuds parviennent à émettre avec succès un message en une étape de temps dépendant de τ. Cela correspond aux hypothèses classiques concernant les canaux multi-accès [14]. Cette hypothèse est justifiée en annexes 3.11.3. Nous supposons également qu'il existe une constante Δ connue, telle que pour tout nœud u, $\delta(u) \leq \Delta$. Ceci peut être vérifié par un contrôle de topologie qui est en mesure d'ajuster la portée de communication ou la puissance de transmission des nœuds lorsque le réseau est trop dense.

Notation. Nous décrivons les algorithmes sous la forme de règles gardées. $G \rightarrow S$ représente une telle règle, où G est un prédicat sur les variables locales d'un nœud, S une affectation de ces mêmes variables locales. Si le prédicat G (la garde) est vrai, l'affectation S est exécutée, sinon elle est ignorée. Certaines gardes peuvent être déclenchées sur événement, par exemple lors de la réception d'un message. Nous supposons que ces événements s'exécutent de manière atomique lors de la réception d'un message. Pour toute configuration du système, quand une garde G est vraie, G est dite *activable* dans cette configuration. L'opérateur $[]$ correspond à la composition non-déterministe des règles gardées ; $([]q : \ q \in M_p : \ G_q \rightarrow S_q)$ est une formulation réduite de l'expression $G_{q1} \rightarrow S_{q_1} \ [] \ G_{q_2} \rightarrow S_{q_2} \ [] \ \cdots \ [] \ G_{q_k} \rightarrow S_{q_k}$, où $M_p = \{q_1, q_2, \ldots, q_k\}$.

Sémantique de l'exécution. L'exécution du système consiste pour chaque nœud à évaluer périodiquement ses règles gardées. Nous supposons que chaque règle *activable* est exécutée en un temps constant (ou ignorée si la règle n'est pas *activable*). De manière générale, nous considérons que lorsqu'un nœud exécute son programme, toutes ses règles *activables* sont effectivement exécutées en un temps constant (par exemple en suivant le modèle du tourniquet).

Propagation des variables partagées. Certaines variables des nœuds sont dites *partagées*. Suivant le schéma présenté en [38], les nœuds diffusent périodiquement les valeurs de leurs variables partagées. Cela signifie que lorsqu'un nœud affecte une valeur à une variable partagée, nous supposons que cette instruction est transformée de telle sorte que, d'une part la variable partagée est régulièrement transmise au voisinage du nœud, et que d'autre part cette retransmission s'effectue de manière probabiliste pour éviter les collisions. Une implantation possible peut être trouvée dans [38]. Dans la suite, nous supposons également que le schéma de [38] est utilisé pour obtenir $\Gamma(u)$ et $\Gamma_2(u)$ pour chaque nœud u.

3.8.2 Construction d'un DAG de hauteur constante

Un DAG ou Directed Acyclic Graph est un graphe simple orienté et sans boucle. Dans notre algorithme, comme dans tout algorithme utilisant l'identifiant des nœuds comme critère de décision finale sans contrainte sur le rayon du cluster (comme dans DDR), le pire cas en terme de stabilisation et de formation de clusters se rencontre quand tous les nœuds ont la même valeur de décision (comme le degré ou la densité) et que les identifiants des nœuds sont uniques et mal distribués. L'algorithme peut alors ne construire qu'un seul cluster dont le diamètre est aussi grand que celui du réseau, le temps de stabilisation dépendant de ce diamètre. De plus, il est évident que construire un tel cluster est inutile puisque nous pourrions tout aussi bien utiliser directement le réseau. Pour pallier cet inconvénient, il peut s'avérer utile d'allouer une couleur aux nœuds, couleur choisie dans un espace Ω constant et plus petit que celui des identifiants, de façon à ce que les couleurs soient localement uniques (dans notre cas, les couleurs doivent être uniques à distance 2 pour qu'un nœud puisse choisir entre deux voisins en compétition) et d'utiliser ces couleurs comme critère de décision finale. Un DAG peut alors être construit à partir de ces couleurs en orientant les arêtes entre les voisins de la couleur la plus grande vers la plus petite.

Notre construction de DAG à hauteur constante est basée sur la technique aléatoire décrite dans [38], mais utilise un espace de couleur beaucoup plus petit Ω ($|\Omega| = \Delta^6$ dans [38], tandis que Δ^2 ou même Δ est suffisant dans notre cas avec $\Delta = max_{u \in V}\delta(u)$).

Soit $Color_u \in \Omega$ une variable partagée désignant la couleur du nœud u. Soit $Color_{\Gamma(u)} = \{ \bowtie Color_v \mid v \in \Gamma(u)\}$, où $\bowtie Color_v$ réfère à la copie en cache de la variable partagée $Color_v$ au nœud u. En d'autres termes, $Color_v$ correspond à la couleur que u pense que v a.

Supposons que $\mathsf{random}(S)$ choisit avec une probabilité uniforme un élément dans un ensemble S. Le nœud u utilise la fonction suivante pour calculer $Color_u$:

$$\mathsf{newColor}(Color_u) = \begin{cases} \bowtie Color_u & \text{si } \bowtie Color_u \notin Color_{\Gamma(u)} \\ \mathsf{random}(\Omega \setminus Color_{\Gamma(u)}) & \text{sinon} \end{cases}$$

L'algorithme de construction d'un DAG à hauteur constante est le suivant :

$$\mathsf{N1} : \quad \mathit{VRAI} \rightarrow Color_u := \mathsf{newColor}(Color_u)$$

Théorème 1 *L'algorithme N1 stabilise avec probabilité* 1 *en un temps constant vers un DAG de hauteur inférieure ou égale à* $|\Omega| + 1$.

Preuve 1 *La preuve de ce théorème est similaire à celle de [38]. Supposons que la hauteur du DAG soit supérieure à* $|\Omega| + 1$. *Cela signifie qu'il existe au moins deux nœuds de même couleur sur une branche du DAG (sur un chemin reliant la racine à une feuille). Or, les arêtes du DAG sont orientées en fonction des couleurs des nœuds qui sont ordonnées. Si sur la même branche, il existe deux nœuds* u *et* v *de même couleur, cela implique que* $Color_u < Color_v$, *ce qui contredit l'hypothèse d'ordre total des couleurs.* ■

3.8.3 Analyse de la construction du DAG de couleurs

Nous avons cherché à caractériser le DAG que nous construisons et son coût. Pour cela, nous avons analysé analytiquement et par simulation le temps de construction du DAG qui correspond au temps de stabilisation de l'algorithme de coloriage. Nous avons également mesuré par simulation l'influence de la taille du domaine des couleurs Ω sur ce temps de stabilisation et sur la taille du DAG résultant. Comme nous allons le voir, il en ressort qu'un compromis est à faire pour déterminer le paramètre Ω : plus la valeur de $|\Omega|$ est grande, plus le temps de convergence de N1 est faible mais plus la hauteur du DAG est importante. Une hauteur de DAG importante augmente le temps de stabilisation des algorithmes qui se basent sur ces DAG.

Analyse théorique du temps de convergence.

Le temps de convergence de l'algorithme de coloriage N1 correspond au nombre d'étapes nécessaires avant que chaque nœud ait une couleur unique dans son voisinage. Pour mener cette étude théorique, nous nous sommes inspirés du protocole NAP [21].

Nous modélisons l'algorithme de coloriage par des lancés successifs de boules dans des urnes. L'ensemble des couleurs est représenté par M urnes dans lesquelles L boules représentant les nœuds sont distribuées.

L'algorithme de coloriage peut être modélisé de la façon suivante en termes d'urnes et de boules :

Algorithm 2 COLORIAGE(L, M)

▷ *Entrées : M urnes et L boules*
▷ *Pré-condition : $M \geq L$*
if $(L \neq 0)$ **then**
  Lance aléatoirement L boules dans les M urnes ;
  Met de côté toutes les urnes contenant exactement une boule avec leur boule ;
  Soit $c \leq M$ le nombre d'urnes isolées ;
  Appelle COLORIAGE($L - c$, $M - c$) ;
end

On remarque qu'une telle analyse ne considère que des graphes complets. Dans un réseau sans fil qui n'est pas nécessairement un graphe complet, deux nœuds voisins (A et B) n'étant pas en conflit mutuel peuvent tout de même tirer une nouvelle couleur simultanément s'ils sont chacun en conflit avec un autre de leur voisin non visible par A ou B, ce qui n'est pas considéré dans cette analyse. Ainsi, l'étude suivante nous fournit une borne inférieure sur le temps de stabilisation de l'algorithme. Cet aspect est plus détaillé dans [55].

Dans chaque voisinage, le but est alors de n'avoir qu'une seule boule (un nœud) associée à une seule urne donnée (une couleur). Soit la variable aléatoire N représentant le nombre d'itérations nécessaires pour obtenir une telle configuration. Le temps de convergence moyen de l'algorithme est l'espérance de N : $\mathbb{E}[N]$. Pour déterminer

$\mathbb{E}[N]$, nous considérons une chaîne de Markov à temps discret $X = \{X_n,\, n \in \mathbb{N}\}$ sur l'espace $\mathcal{I} = 0, 1, ..., L$. $X_n = i$ représente le fait qu'après n transitions, exactement i boules et urnes ont été mises de côté.

Nous notons $\mathcal{P}(L, M) = (p_{i,j}(L, M))_{(i,j)\in\mathcal{I}^2}$ la matrice de probabilité de transition de la chaîne de Markov X. L'espérance $\mathbb{E}[N]$ peut être déduite du calcul des $p_{i,j}(L, M)$. $p_{i,j}(L, M)$ représente la probabilité d'avoir exactement j urnes de côté au temps $n+1$ sachant que i urnes étaient de côté au temps n. $p_{i,j}(L, M)$ peut aussi être vu comme la probabilité d'obtenir exactement $j - i$ urnes avec exactement une boule en lançant $L - i$ boules dans $M - i$ urnes. Nous obtenons donc, pour tout $i \leq j$:

$$p_{i,j}(L, M) = p_{i,j} = p_{0,j-i}(L - i, M - i). \tag{3.1}$$

X est acyclique et l'état L est un état absorbant. Cela signifie que pour tout $i \in \mathcal{I}-\{L\}$ et tout $j \in \mathcal{I}$, $p_{i,j}(L, M) = 0$ si $i > j$ et $p_{L,L}(L, M) = 1$.

Grâce à la relation 3.1, nous pouvons ne calculer que les valeurs des $p_{0,j-i}(L-i, M-i)$ pour $i \leq j$ pour obtenir toutes les valeurs de la matrice $\mathcal{P}(L, M)$.

$p_{0,j}(L, M)$ est la probabilité d'obtenir exactement j urnes avec exactement une boule lors du lancer de L boules dans M urnes. Le cas $j = L$ conduit au problème des anniversaires, d'où :

$$p_{0,L}(L, M) = \frac{M!}{(M - L)!M^L}$$

Pour $j < L$, nous procédons de la sorte. L boules sont lancées dans M urnes. On note $K_0(L, M)$ le nombre d'urnes vides et $K_1(L, M)$ le nombre d'urnes contenant exactement une boule. Soit $a_{L,M}(k, j)$ la distribution jointe des deux variables aléatoires K_0 et K_1 :

$$a_{L,M}(k, j) = \mathbb{P}[K_0(L, M) = k,\, K_1(L, M) = j]$$

Les $p_{0,j}(L, M)$ peuvent alors s'écrire :

$$p_{0,j}(L, M) = \mathbb{P}[K_1(L, M) = j] = \sum_{k=0}^{M} a_{L,M}(k, j)$$

Ainsi, afin de calculer la matrice de transition $\mathcal{P}(L, M)$, il ne nous reste qu'à déterminer les $a_{L,M}(k, j)$. Pour cela, nous raisonnons par récurrence en conditionnant le résultat du dernier lancer : pour obtenir k urnes vides et j urnes avec exactement une boule en lançant L boules dans M urnes, il faut qu'à la fin du lancer de la $L - 1^{ieme}$ balle :

1. soit avoir $k + 1$ urnes vides et $j - 1$ urnes avec exactement une boule et lancer la dernière boule dans une urne vide ;
2. soit avoir k urnes vides et $j + 1$ urnes avec exactement une boule et lancer la dernière boule dans une urne qui contenait exactement une boule ;
3. soit avoir k urnes vides et j urnes avec exactement une boule et lancer la dernière boule dans une urne qui contenait au moins deux boules.

Pour $L \geq 2$, on obtient :

$$a_{L,M}(k,j) = \frac{k+1}{M} a_{L-1,M}(k+1, j-1) 1_{\{j \geq 1\}} + \frac{j+1}{M} a_{L-1,M}(k, j+1) + \frac{M-(j+k)}{M} a_{L-1,M}(k,j)$$

où $1_{\{c\}} = 1$ si la condition c est remplie et 0 sinon.

Les $a_{L,M}(k,j)$ peuvent être calculés par récursion en considérant que si $L = 1$:

$$a_{1,M}(k,j) = 1_{\{k=M-1,\, j=1\}}$$

On remarque aussi que $a_{L,M}(k,j) = 0$ si $j > L$, si $k = M$ ou si $j+k > M$. De même, si $j = L$, on a $a_{L,M}(k,L) = 0$ pour $k \neq M - L$ et $a_{L,M}(M-L, L) = p_{0,L}(L,M)$.

Une fois la matrice de probabilités de transition obtenue, on peut déterminer la distribution de N ($\mathbb{P}[N = n]$ pour $n = 0, \ldots, \infty$) et sa valeur moyenne $\mathbb{E}[N]$. Les calculs sont les mêmes que ceux menés dans l'étude du protocole NAP. Ils dérivent des résultats classiques des chaînes de Markov. Nous les utilisons directement ici.

Nous définissons Q la sous-matrice obtenue à partir de $\mathcal{P}(L,M)$, en retirant la dernière ligne et la dernière colonne qui correspondent à l'état absorbant L.

Soit α le vecteur ligne contenant la distribution initiale des probabilités des états transitoires de X. α est tel que $\alpha = (\mathbb{P}[X_0 = i])_{i=0,\ldots,L-1}$. La chaîne de Markov X commence en l'état 0 avec la probabilité 1, d'où $\alpha = (1, 0, \ldots, 0)$.

De par les résultats classiques des chaînes de Markov, on obtient :

$$\mathbb{P}[N = n] = \alpha Q^{n-1}(I - Q)\mathbb{1}, \text{ pour } n \geq 1,$$

$$\mathbb{P}[N > n] = \sum_{k=n+1}^{\infty} \alpha Q^{k-1}(I-Q)\mathbb{1} = \alpha Q^n \mathbb{1}, \text{ pour } n \geq 0,$$

$$\mathbb{E}[N] = \sum_{n=0}^{\infty} \mathbb{P}[N > n] = \alpha (I - Q)^{-1} \mathbb{1}$$

où I est la matrice identité et $\mathbb{1}$ le vecteur colonne unité, tous deux de dimension L.

On note $V = (V_i)_{0 \leq i \leq L-1}$ le vecteur d'espérance conditionnelle défini par $V_i = \mathbb{E}[N | X_0 = i]$. On a $V = (I-Q)^{-1}\mathbb{1}$. D'où $\mathbb{E}[N] = \alpha V = V_0$. Le vecteur V est solution du système linéaire $(I-Q)V = \mathbb{1}$. Cela peut s'écrire également $V = \mathbb{1} + QV$. Ainsi, comme la matrice $\mathcal{P}(L,M)$ est acyclique, on obtient, pour $i = L-2 \ldots, 0$:

$$V_i = \frac{1}{1 - p_{i,i}} \left(1 + \sum_{j=i+1}^{L-1} p_{i,j} V_j \right)$$

Comme $V_{L-1,L-1} = 1/(1 - p_{L-1,L-1})$, on peut obtenir V_0 récursivement, et ainsi, obtenir le temps de convergence moyen $\mathbb{E}[N]$.

Simulations.

Afin de valider nos résultats théoriques et de mesurer l'impact de la taille du domaine des couleurs Ω sur le temps de convergence et sur la hauteur du DAG, nous avons mené des simulations utilisant plusieurs valeurs de $|\Omega|$. Nous avons simulé l'Algorithme N1 en utilisant $|\Omega| = (max_{p\in V}|\Gamma(p)|)^2$, cette valeur étant celle considérée par certains algorithmes auto-stabilisants basés sur le coloriage [38]. Nous avons également étudié l'Algorithme N1 en utilisant $|\Omega| = 2 \times (max_{p\in V}|\Gamma(p)|)$. De plus, comme dans les réseaux sans fil, les nœuds n'ont aucune connaissance globale du réseau et donc, aucun moyen *a priori* de connaître le plus fort degré du graphe, nous avons également mené des simulations en utilisant une taille de domaine de couleur propre à chaque nœud, telle que : $\forall p \in V$, $|\Omega_p| = (|\Gamma(p)|)^2$.

Nous avons alors considéré le temps de convergence et la taille du DAG induit.

Dans un premier temps, nous avons comparé les résultats théoriques et de simulation afin de valider chaque approche. La table 3.10 montre les temps de convergence de l'algorithme de coloriage à distance 1 sur une topologie grille où chaque nœud interne a respectivement 4 ou 8 voisins. On remarque que les résultats s'accordent.

4 voisins				8 voisins			
$2 * Max$		Max^2		$2 * Max$		Max^2	
Théorie	Simulation	Théorie	Simulation	Théorie	Simulation	Théorie	Simulation
2.14	2.14	1.56	1.61	1.56	1.67	1.15	1.21

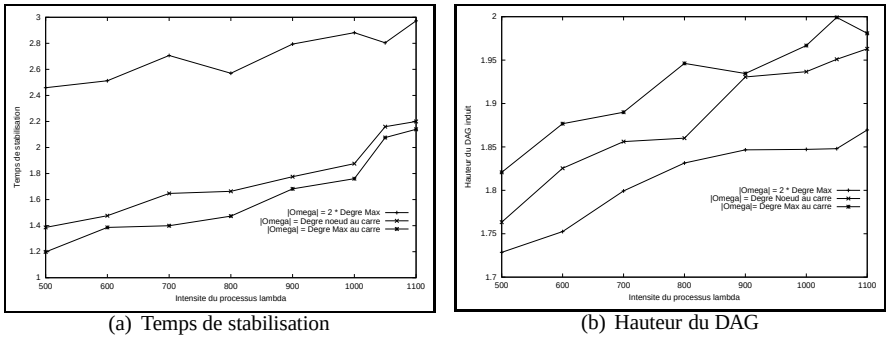
TAB. 3.10 – Temps de stabilisation théorique et obtenu par simulation avec $|\Omega| = (max_{p\in V}|N_p|)^2$) et $|\Omega| = 2 \times (max_{p\in V}|N_p|)$ dans une grille à 4 et 8 voisins.

La figure 3.14 montre l'influence de la taille du domaine sur le temps de convergence et la hauteur du DAG dans le cas d'un coloriage à distance 1. Les résultats montrent clairement que plus la valeur de $|\Omega|$ est grande, plus le temps de convergence de N1 est faible mais plus la hauteur du DAG est importante. Il y a donc un compromis à faire pour déterminer le paramètre Ω.

3.8.4 Utilisation des couleurs pour le *clustering*

Dans cette section, nous ré-écrivons l'algorithme de *clustering* avec les règles d'auto-stabilisation. Chaque nœud u maintient deux variables partagées : $\rho(u)$ et $\mathcal{H}(u)$ où $\rho(u)$ est la densité du nœud u et $\mathcal{H}(u)$ son cluster-head.

Afin d'utiliser le DAG des couleurs dans l'algorithme de *clustering*, nous redéfinissons l'opérateur d'ordre binaire $\prec$ défini dans la section 3.2 de la façon suivante : pour $(u, v) \in V^2$, $u \prec v$ si et seulement si $\{\rho(u) < \rho(v)\}$ ou $\{\rho(u) = \rho(v) \wedge Age(u) < Age(v)\}$ ou $\{\rho(u) = \rho(v) \wedge Age(u) = Age(v) \wedge \mathit{Color}_v < \mathit{Color}_u\}$. Soit $\max_{\prec}$ la fonction de maximum associée à l'opérateur d'ordre binaire $\prec$. Quand un nœud u calcule le



(a) Temps de stabilisation (b) Hauteur du DAG

FIG. 3.14 – Influence de Ω.

résultat de $\prec$ ou de $\max_{\prec}$, il utilise les valeurs cache de son voisinage en supposant $\boxtimes Color_u = Color_u$ et $\boxtimes\rho(u) = \rho(u)$.

Nous définissons maintenant la fonction clusterHead d'élection de cluster-head :

$$\mathsf{clusterHead} = \begin{cases} u & \text{si } \forall v \in \Gamma(u), v \prec u, \\ \mathcal{H}(\max_{\prec}\{v \in \Gamma(u)\}) & \text{sinon.} \end{cases}$$

L'algorithme s'exécute comme suit :

$$\begin{array}{ll} \mathsf{R1}: & \mathit{VRAI} \rightarrow \rho(u) := \mathsf{densite} \\ \mathsf{R2}: & \mathit{VRAI} \rightarrow \mathcal{H}(u) := \mathsf{clusterHead} \end{array}$$

Lemme 4 *Partant de n'importe quelle configuration initiale, chaque nœud u a une valeur de densité* correcte $\rho(u)$ *en un temps borné constant.*

Preuve 2 *Après un temps constant, chaque nœud u a une vue correcte de son 2-voisinage. Puis, après l'exécution de la règle R1, la densité $\rho(u)$ du nœud u est correcte.* ■

Lemme 5 *Partant de n'importe quelle configuration initiale, chaque nœud u a une valeur* correcte *pour $\mathcal{H}(u)$ en un temps borné constant.*

Preuve 3 *Supposons que tout nœud a une valeur correcte de sa densité (vrai après un temps constant d'après le Lemme 4). Après que la variable partagée $\rho(u)$ ait été communiquée sans collision à tout nœud de $\Gamma(u)$ (cela arrive après un temps constant), chaque nœud a une valeur cache correcte pour la densité de chacun de ses voisins. Nous considérons maintenant le DAG induit par la relation $\prec$ (noté $DAG_{\prec}$ par la suite). En un temps constant, les racines de $DAG_{\prec}$ ont une valeur correcte de l'identité de leur cluster-head (puisqu'il s'agit de leur propre identifiant). Supposons que tout nœud à distance inférieure ou égale à n des racines de $DAG_{\prec}$ a une valeur correcte de l'identité de leur cluster-head. Sur exécution de la règle R2 sur les nœuds à distance*

$n+1$ des racines de DAG$_\prec$, ces nœuds obtiennent alors une valeur correcte de l'identité de leur cluster-head (puisque le cluster-head est déterminé de façon déterministe (i) *par la densité et la topologie locale – qui est fixe – et* (ii) *par l'identité des cluster-heads des nœuds à distance inférieure ou égale à n des racines de DAG$_\prec$). Par induction, le temps nécessaire à la stabilisation de l'algorithme est proportionnel à la hauteur du DAG$_\prec$.*

Nous prouvons maintenant que la hauteur du DAG$_\prec$ est bornée par une constante. Les couleurs des nœuds sont bornées par une constante $|\Omega|$. Le nombre d'arêtes dans le 1-voisinage d'un nœud est bornée par Δ^2, le nombre de 1-voisins est borné par Δ, d'où, le nombre de valeurs possibles pour la densité est au plus de Δ^3. Le nombre de couples (densité, couleur) possibles pour un nœud est $|\Omega|\Delta^3$, lui-même borné par une constante. Ainsi, la hauteur du DAG$_\prec$ est lui-aussi borné par une constante.

L'algorithme stabilise en un temps proportionnel à la hauteur du DAG$_\prec$, celle-ci étant constante. Donc le temps de stabilisation est lui aussi borné par une constante. ■

3.8.5 Validation des propriétés auto-stabilisantes

Comme mentionné dans la section 3.8.1, nous supposons l'existence d'une constante $\tau > 0$ telle que chaque nœud est en mesure de diffuser localement une trame et d'en recevoir une de chacun de ses voisins en un temps borné, appelé une *étape de temps*. Après une étape, chaque nœud connaît ses 1-voisins. Après deux étapes, il connaît ses 2-voisins et peut calculer sa valeur de densité et après trois étapes, il connaît son père. Le nombre d'étapes nécessaires à un nœud pour connaître l'identité de son cluster-head dépend directement de la distance qui l'en sépare et est borné par la hauteur de l'arbre auquel il appartient.

Les simulations menées ici nous ont permis d'évaluer l'importance de l'introduction des couleurs. Le modèle de simulation est toujours celui décrit dans le chapitre 1.1. Les nœuds sont déployés suivant un Processus de Points de Poisson avec différentes valeurs de λ et de R.

L'allocation des couleurs se fait suivant l'Algorithme N1. Chaque nœud se choisit aléatoirement une couleur entre 0 et $|\Omega| = \Delta^2$ (avec $\Delta = max_{v \in V}\delta(u)$). Il compare alors sa couleur à celle de ses voisins. Si deux voisins ont la même couleur, le nœud dont la couleur est la plus petite se choisit une autre couleur et ainsi de suite jusqu'à ce qu'il n'existe aucune paire de nœuds voisins portant la même couleur. À partir de là, les clusters sont construits suivant l'algorithme 1 en utilisant les couleurs comme critère de décision finale.

Les caractéristiques des clusters obtenus sont données dans la table 3.11 pour $\lambda = 1000$ et différentes valeurs de R. Bien que donnés pour $\lambda = 1000$, les résultats sont similaires quelle que soit la valeur de λ.

On remarque que quel que soit R (et donc le degré δ des nœuds), l'excentricité moyenne des cluster-heads et la hauteur des arbres varient peu. Cela confirme notre hypothèse stipulant que la transmission de l'identité du chef de cluster se fait en un

temps constant. On notera également que dans un tel cas où les densités et identifiants des nœuds sont uniformément distribués, l'utilisation des couleurs n'apporte pas grand chose. Cela est dû au fait que dans une telle distribution, les nœuds utilisent uniquement les densités pour déterminer leur père puisqu'elles sont rarement égales.

	$R = 0.05\ (\overline{\delta} = 7.85)$		$R = 0.08\ (\overline{\delta} = 20.11)$		$R = 0.1\ (\overline{\delta} = 31.42)$	
Couleurs	avec	sans	avec	sans	avec	sans
Nb clusters	61.0	61.4	19.2	19.5	11.7	11.7
$\tilde{e}(\mathcal{H}(u)/\mathcal{C}(u))$	2.6	2.6	3.1	3.1	3.2	3.2
Hauteur arbre-cluster	2.7	2.7	3.3	3.3	3.5	3.5

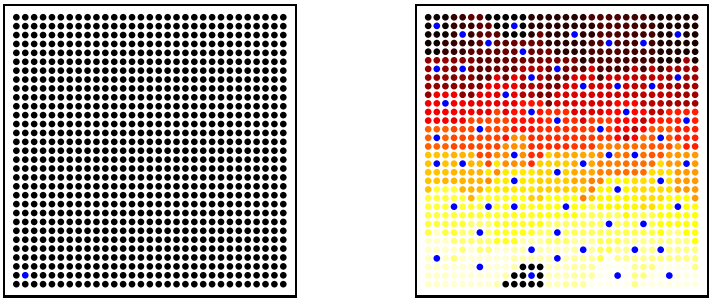
TAB. 3.11 – Caractéristiques des clusters pour une topologie géométrique aléatoire avec $\lambda = 1000$.

Considérons maintenant un scénario où les nœuds sont distribués dans une grille avec des identifiants allant croissant de la gauche vers la droite et du bas vers le haut. Dans ce cas, tous les nœuds intérieurs de la grille ont la même valeur de densité et le même degré. Le seul moyen de choisir leur père est d'utiliser les identifiants. Comme ceux-ci sont mal répartis, tous les nœuds vont finalement s'attacher au même cluster-head, comme le montre le tableau 3.12. Dans un pareil cas, on remarquera que l'introduction des couleurs est utile car elle permet de réduire de façon drastique le nombre d'étapes nécessaires avant la stabilisation (puisqu'elle réduit fortement la hauteur des arbres de *clustering*) et de construire des clusters plus adaptés. La figure 3.15 montre un exemple de clusters obtenus pour $R = 0.05$. Les cluster-heads apparaissent en bleu, une couleur par cluster. Sur la figure 3.15(a), les couleurs ne sont pas utilisées et seulement un cluster est créé. Sur la figure 3.15(b), les couleurs sont considérées et plusieurs clusters homogènes sont créés.

	Grille 32×32		Grille 18×18		Grille 15×15	
Couleurs	avec	sans	avec	sans	avec	sans
Nb clusters	52.8	1.0	29.3	1.0	18.5	1.0
$\tilde{e}(\mathcal{H}(u)/\mathcal{C}(u))$	3.4	29.1	4.1	19.1	3.6	6.5
Hauteur arbres-cluster	3.7	83.4	4.7	100.5	4.5	32.1

TAB. 3.12 – Caractéristiques des clusters formés sur une grille à 8 voisins.

Dans cette partie, nous avons introduit un mécanisme supplémentaire dans la construction des clusters qui permet à notre algorithme de se stabiliser en un temps rapide et borné et ce, quelle que soit la topologie sous-jacente. Ce caractère local d'auto-stabilisation apporte une stabilité à notre algorithme et une robustesse face aux pannes et attaques. En effet, lorsqu'un tel phénomène survient, l'auto-stabilisation permet au réseau de ne pas être impacté dans son ensemble par la cassure de liens. Les nœuds sont capables d'isoler la faute et de la réparer.



(a) sans utiliser les couleurs (b) en considérant les couleurs

FIG. 3.15 – Exemple de constructions obtenues pour des grilles à 8 voisins.

3.9 Conclusion

Dans ce chapitre, nous avons introduit une nouvelle métrique qui permet d'organiser un réseau sans fil multi-sauts en clusters. Nous avons ensuite analysé cette métrique analytiquement et par simulation, ainsi que la structure de clusters qu'elle permet de construire. L'algorithme de *clustering* et le calcul de cette métrique sont locaux, distribués et ne nécessitent la connaissance que du voisinage à deux sauts pour chaque nœud. Ils sont donc peu coûteux et permettent une maintenance locale donc rapide. L'algorithme de *clustering* ne repose sur aucun paramètre fixé *a priori* et a été prouvé auto-stabilisant en un temps borné et constant. La structure de clusters formée présente d'intéressantes caractéristiques. Comparée à d'autres algorithmes de la littérature, elle s'avère plus robuste face à la mobilité des nœuds et s'adapte mieux à la topologie sous-jacente.

Tout réseau doit permettre aux entités de communiquer et nécessite pour cela un protocole de routage/localisation et un processus de diffusion de messages. Grâce aux caractéristiques de notre structure de clusters que nous avons dégagées au travers de nos études et analyses, nous avons pu proposer deux utilisations de la structure qui tirent avantage de ces propriétés : un processus de diffusion (chapitre 4) et un processus de localisation et de routage (chapitre 5) pour permettre aux entités du réseau de communiquer.

3.10 Publications

1. **Colloques et conférences internationaux avec comité de lecture :**
   (a) *Self-stabilization in self-organized Multihop Wireless Networks*. Nathalie Mitton, Éric Fleury, Isabelle Guérin-Lassous and Sébastien Tixeuil. Workshop on Wireless Ad Hoc Networking (WWAN'05), Juin 2005, Columbus, Ohio, USA.
   (b) *Self-organization in large scale ad hoc networks*. Nathalie Mitton, Anthony Busson and Éric Fleury. Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET'04), Juin 2004, Bodrum, Turquie.

2. **Colloques et conférences nationaux :**
   (a) *Auto-stabilisation dans les réseaux ad hoc*. Nathalie Mitton, Éric Fleury, Isabelle Guérin-Lassous et Sébastien Tixeuil. ALGOTEL'05, Mai 2005, Presqu'île de Giens, France.
   (b) *Auto-organisation dans les réseaux ad-hoc à grandes échelles*. Nathalie Mitton, Anthony Busson et Éric Fleury. ALGOTEL'04, Mai 2004, Batz-sur-mer, France.
   (c) *Auto-organisation dans les réseaux ad-hoc à grandes échelles*. Nathalie Mitton et Éric Fleury. Journées Graphes Réseaux et Modélisation, GRM'03, Décembre 2003, Paris, France.

3. **Rapports de recherche :**
   (a) *On Fast Randomized Colorings in Sensor Networks*. Nathalie Mitton, Éric Fleury, Isabelle Guérin-Lassous and Bruno Séricola and Sébastien Tixeuil. LRI-1416. Juin 2005.
   (b) *Self-stabilization in self-organized Multihop Wireless Networks*. Nathalie Mitton, Éric Fleury, Isabelle Guérin-Lassous and Sébastien Tixeuil. RR-5426. Décembre 2004.
   (c) *Analysis of the Self - organization in Multi-hops wireless networks*. Nathalie Mitton, Anthony Busson and Éric Fleury. RR-5328. Octobre 2004.
   (d) *Self-organization in large scale ad hoc networks*. Nathalie Mitton and Éric Fleury. RR-5042. Décembre 2003.

4. **Séminaires, présentations, exposés :**
   (a) *Auto-organisation dans les réseaux* ad hoc *grandes échelles.* Nathalie Mitton, Éric Fleury. Séminaire ACI Pair à Pair - Arcachon - France - 6-7 Mai 2004.

3.11 Annexes

3.11.1 Analyse de la densité moyenne

Nous donnons ici la preuve du lemme 1 qui donne la valeur moyenne de la 1-densité d'un nœud :

Lemme 1 La 1-densité moyenne de tout nœud u est :

$$\tilde{\rho}(u) = \mathbb{E}^o\left[\rho(0)\right] = 1 + \frac{1}{2}\left(\pi - \frac{3\sqrt{3}}{4}\right)\left(\lambda R^2 - \frac{1 - exp\{-\lambda\pi R^2\}}{\pi}\right)$$

Preuve 4 *Soit $(Y_i)_{i=1,..,\Phi(B'_0)}$, chacun des points de Φ se trouvant dans B'_0. Par définition de la densité, on a :*

$$\mathbb{E}^o\left[\rho(0)\right] = 1 + \frac{1}{2}\mathbb{E}^o\left[\sum_{i=1}^{\Phi(B'_0)} \frac{\Phi(B'_0 \cap B'_{Y_i})}{\Phi(B'_0)}\right]$$

B'_u est la boule centrée en u de rayon R, privée du singleton $\{u\}$: $B'_u = B(u,R)\backslash\{u\}$. $\Phi(B'_0 \cap B'_{Y_i})$ correspond au nombre de voisins communs aux nœuds 0 et Y_i. En faisant ainsi la somme des voisins communs à 0 et Y_i pour tous les Y_i, on obtient le nombre de liens entre les voisins de 0. Cependant, chaque lien est ainsi compté deux fois (un lien entre Y_i et Y_j est compté quand on considère Y_i voisin commun de Y_j et 0 et quand on considère Y_j voisin commun de 0 et Y_i). C'est pourquoi on se doit de diviser cette somme par 2.

Nous supposons que $\rho(0) = 1$ si le nœud 0 n'a aucun voisin ($\Phi(B'_0) = 0$). Nous conditionnons par la valeur du degré du nœud 0 : $\delta(0) = \Phi(B'_0)$. Nous obtenons alors :

$$\begin{aligned}
\mathbb{E}^o\left[\rho(0)\right] &= \mathbb{E}^o\left[\rho_0|\delta_0 = 0\right]\mathbb{P}^o(\delta_0 = 0) + \sum_{k=1}^{+\infty}\mathbb{E}^o\left[\rho_0|\delta_0 = k\right]\mathbb{P}^o(\delta_0 = k) \\
&= 1 + \frac{1}{2}\sum_{k=1}^{+\infty}\mathbb{E}^o\left[\sum_{i=1}^{\Phi(B'_0)} \frac{\Phi(B'_0 \cap B'_{Y_i})}{\Phi(B'_0)}\Big|\Phi(B'_0) = k\right]\mathbb{P}^o\left(\Phi(B'_0) = k\right) \\
&= 1 + \frac{1}{2}\sum_{k=1}^{+\infty}\sum_{i=1}^{k}\frac{1}{k}\mathbb{E}^o\left[\Phi(B'_0 \cap B'_{Y_i})\Big|\Phi(B'_0) = k\right] \times \mathbb{P}^o\left(\Phi(B'_0) = k\right)
\end{aligned} \tag{3.2}$$

Les nœuds $(Y_i)_{i=1,..,k}$ étant indépendants et uniformément distribués dans B'_0, l'espérance $\mathbb{E}^o\left[\Phi(B'_0 \cap B'_{Y_i})\Big|\Phi(B'_0) = k\right]$ est la même pour tout $i, i = 1,..,k$

(et donc pour tout voisin Y_i de 0). On note $\nu(S)$ la mesure de Lebesgue de la région du plan S dans $\mathbb{R}^2$. Connaissant $\nu(B'_0 \cap B'_{Y_i})$ et sachant que $\Phi(B'_0) = k$, alors le nombre de nœuds dans $B'_0 \cap B'_{Y_i}$ suit une loi binomiale de paramètre $\left(k-1, \frac{\nu(B'_0 \cap B'_{Y_i})}{\nu(B'_0)}\right)$.

Le nombre moyen de points devient : $(k-1)\frac{\nu(B'_0 \cap B'_{Y_i})}{\nu(B'_0)}$.

D'où, pour tout $i = 1, .., k$:

$$\begin{aligned}\mathbb{E}^o\left[\Phi(B'_0 \cap B'_{Y_i})\middle|\Phi(B'_0) = k\right] =& \frac{(k-1)}{\pi R^2}\mathbb{E}^o\left[\nu(B'_0 \cap B'_{Y_i})\middle|\Phi(B'_0) = k\right] \\ =& \frac{(k-1)}{\pi R^2}\mathbb{E}^o\left[\nu(B'_0 \cap B'_{Y_i})\right]\end{aligned} \tag{3.3}$$

Cette égalité 3.3 vient du fait que la surface $\nu(B'_0 \cap B'_{Y_i})$ ne dépend pas du nombre de Y_i, puisque les nœuds Y_i sont indépendants.

Si on pose que le nœud Y_i est à une distance r du nœud 0, l'aire de l'intersection $B'_0 \cap B'_{Y_i}$ devient $\nu(B'_0 \cap B'_{Y_i}) = A(r) = 2R^2 arccos\frac{r}{2R} - r\sqrt{R^2 - \frac{r^2}{4}}$, comme illustré sur la figure 3.4.

Puisque les Y_i sont uniformément distribués dans B'_0, la valeur moyenne de l'aire intersection est :

$$\begin{aligned}\mathbb{E}^o\left[\nu(B'_0 \cap B'_{Y_i})\right] &= \mathbb{E}^o\left[A(r)\right] \\ &= \int_0^{2\pi}\int_0^R \frac{A(r)}{\pi R^2}\, r\, dr\, d\theta \\ &= R^2\left(\pi - \frac{3\sqrt{3}}{4}\right)\end{aligned}$$

Soit p la probabilité que deux voisins de 0 soient eux-mêmes voisins. p est la valeur moyenne de l'aire d'intersection divisée par la surface totale où peuvent se trouver les voisins de 0 (πR^2). On a :

$$\begin{aligned}p &= \mathbb{P}\left(Y_2 \in B_0 \cap B_{Y_1}\right) = \frac{2}{\pi}\int_{u=0}^{1}\left(2\arccos\frac{u}{2} - u\sqrt{1 - \frac{u^2}{4}}\right)u du = 1 - \frac{3\sqrt{3}}{4\pi} \\ &\approx 0.5865\end{aligned}$$

Ce résultat combiné à celui de l'équation 3.2 nous donne :

$$\begin{aligned}\mathbb{E}^o\left[\rho(0)\right] =& 1+\frac{1}{2}\sum_{k=1}^{+\infty}\sum_{i=1}^{k}\frac{1}{k}\frac{k-1}{\pi}\left(\pi-\frac{3\sqrt{3}}{4}\right)\mathbb{P}^o\left(\Phi(B'_0)=k\right)\\ =& 1+\frac{1}{2}\sum_{k=1}^{+\infty}\frac{k-1}{\pi}\left(\pi-\frac{3\sqrt{3}}{4}\right)\mathbb{P}^o\left(\Phi(B'_0)=k\right)\\ =& 1+\frac{1}{2\pi}\left(\pi-\frac{3\sqrt{3}}{4}\right)\times\left(\sum_{k=1}^{+\infty}k\mathbb{P}^o\left(\Phi(B'_0)=k\right)-\sum_{k=1}^{+\infty}\mathbb{P}^o\left(\Phi(B'_0)=k\right)\right)\\ =& 1+\frac{1}{2\pi}\left(\pi-\frac{3\sqrt{3}}{4}\right)\left(\lambda\pi R^2-\left(1-exp\{\lambda\pi R^2\}\right)\right)\end{aligned} \tag{3.4}$$

L'égalité 3.4 découle du théorème de Slyvniack dont l'une des conséquences est que le nombre de voisins $\Phi(B_0^{'})$ de 0, sous la probabilité de Palm, suit une loi de Poisson discrète de paramètre $\lambda\pi R^2$. ■

3.11.2 Calcul analytique du nombre de clusters

Nous donnons ici les calculs détaillés de la borne du nombre de clusters donné dans le théorème 1. Nous bornons la probabilité qu'un nœud soit chef.

Conjecture 1 Une borne supérieure pour la probabilité qu'un nœud soit chef est :

$$\mathbb{P}^o_\Phi\left(\rho(0)>\max_{k=1,..,\Phi(B_0)}\rho(Y_k)\right)\leq\left(1+\sum_{n=1}^{+\infty}\frac{1}{n}\frac{\left(\lambda\pi R^2\right)^n}{n!}\right)\exp\left\{-\lambda\pi R^2\right\}$$

Preuve 5 *Calculer la probabilité pour un nœud d'être chef revient à calculer la probabilité pour un nœud d'avoir la plus forte densité dans son voisinage.*

Nous considérons le point 0. Soient $B_0=B(0,R)$ la boule de rayon R centrée en 0 et $B_0^{'}$ la boule de rayon R centrée en 0 privée du singleton $\{0\}$. Soit $(Y_i)_{i=1,..,\Phi(B'_0)}$, chacun des points de Φ se trouvant dans B'_0.

La densité des points de B_0 est équi-distribuée puisque les positions de ces points sont uniformément et indépendamment distribuées dans B_0. D'où :

$$\mathbb{P}^o\left(\rho(Y_i)>\max_{k=1,..,n;k\neq i}\rho(Y_k)\Big|\Phi(B_0^{'})=n\right)\quad\leq\quad\frac{1}{n}$$

Si le point 0 n'a aucun voisin ($\Phi(B_0^{'})=0$), 0 est un cluster-head.

Nous avons :

$$\begin{aligned}&\mathbb{P}^o\left(\rho(0) > \max_{k=1,..,\Phi(B_0^{'})} \rho(Y_k)\right)\\ =&\mathbb{P}^o\left(\rho(0) > \max_{k=1,..,\Phi(B_0^{'})} \rho(Y_k)\Big|\Phi(B_0^{'}) > 0\right)\mathbb{P}^o\left(\Phi(B_0^{'}) > 0\right) + \mathbb{P}^o\left(\Phi(B_0^{'}) = 0\right)\end{aligned}$$

On note :

$$p_0 = \mathbb{P}^o\left(\rho(0) > \max_{k=1,..,\Phi(B_0^{'})} \rho(Y_k)\Big|\Phi(B_0^{'}) > 0\right) \times \mathbb{P}^o\left(\Phi(B_0^{'}) > 0\right)$$

Si nous supposons que les densités sont équi-distribuées, nous avons :

$$p_0 < \mathbb{P}^o\left(\rho(Y_1) > \max(\rho(0), \max_{k=2,..,\Phi(B_0^{'})} \rho(Y_k))\Big|\Phi(B_0^{'}) > 0\right) \times \mathbb{P}^o\left(\Phi(B_0^{'}) > 0\right)$$

Il s'agit d'une conjecture, en effet nous n'avons pas réussi à démontrer ce résultat. Cependant d'après nos simulations, quelle que soit la densité des nœuds, la quantité p_0 est deux à trois fois plus petite que le terme de droite de cette inégalité.

De plus, comme l'évènement

$$E_1 = \{\rho(Y_1) > \max(\rho(0), \max_{k=2,..,\Phi(B_0^{'})} \rho(Y_k))\}$$

est inclus dans l'évènement

$$E_2 = \{\rho(Y_1) > \max_{k=2,..,\Phi(B_0^{'})} \rho(Y_k))\}$$

nous pouvons majorer la probabilité que E_1 se produise par la probabilité que E_2 se réalise. Nous obtenons :

$$\begin{aligned}p_0 &\leq \mathbb{P}^o[E_1] \times \mathbb{P}^o\left(\Phi(B_0^{'}) > 0\right) \leq \mathbb{P}^o[E_2] \times \mathbb{P}^o\left(\Phi(B_0^{'}) > 0\right)\\ p_0 &\leq \mathbb{P}^o\left(\rho(Y_1) > \max_{k=2,..,\Phi(B_0^{'})} \rho(Y_k)\Big|\Phi(B_0^{'}) > 0\right) \times \mathbb{P}^o\left(\Phi(B_0^{'}) > 0\right)\\ &= \sum_{n=1}^{+\infty} \mathbb{P}^o\left(\rho(Y_1) > \max_{k=2,..,\Phi(B_0^{'})} \rho(Y_k)\Big|\Phi(B_0^{'}) = n\right) \times \mathbb{P}^o\left(\Phi(B'_0) = n\right)\\ &\leq \sum_{n=1}^{+\infty} \frac{1}{n} \frac{\left(\lambda\pi R^2\right)^n}{n!} \exp\left\{-\lambda\pi R^2\right\}\end{aligned}$$

De plus, d'après le théorème de Slivnyak [76], le nombre de points sous la distribution de Palm dans un espace Borélien de $\mathbb{R}^2$ qui ne contient pas le point 0, suit une loi de Poisson discrète. Nous en déduisons :

$$\mathbb{P}^o\left(\rho(0) > \max_{k=1,..,\Phi(B_0)} \rho(Y_k)\right) \leq \exp\left\{-\lambda\pi R^2\right\} + \sum_{n=1}^{+\infty} \frac{1}{n} \frac{\left(\lambda\pi R^2\right)^n}{n!} \exp\left\{-\lambda\pi R^2\right\}$$

■

Comme, d'après le lemme 3, le nombre de clusters est tel que $\mathbb{E}\left[\text{Nb de clusters dans } C\right] = \lambda\nu(C)\mathbb{P}^o_\Phi\left(0 \text{ est chef }\right)$*, on obtient une borne supérieure pour le nombre de clusters formés par notre algorithme dans une surface C :*

$$\mathbb{E}\left[\text{Nb de clusters dans } C\right] \leq \lambda\nu(C) \exp\left\{-\lambda\pi R^2\right\} + \sum_{n=1}^{+\infty} \frac{1}{n} \frac{\left(\lambda\pi R^2\right)^n}{n!} \exp\left\{-\lambda\pi R^2\right\}$$

3.11.3 Temps de transmission borné

Dans cette partie, nous justifions l'hypothèse suivante : *"il existe une constante $\tau > 0$ telle que chaque nœud est en mesure de diffuser localement une trame et d'en recevoir une de chacun de ses voisins en un temps borné $\Delta(\tau)$"* faite dans la section 3.8.5 pour prouver le caractère d'auto-stabilisation de notre algorithme.

Dans [83], les auteurs fournissent une analyse des performances du protocole IEEE 802.11 pour la couche MAC des réseaux sans fil. En considérant un graphe à n stations, toutes à portée de transmission les unes des autres (*c.à.d.* que le graphe de communication est complet), les auteurs modélisent l'activation de la période de contention par les nœuds avant l'émission d'une trame. La durée de cette période dépend des collisions qui ont pu se produire pour cette trame auparavant. On trouve en particulier dans ce papier la probabilité P_{suc} qu'il y ait une transmission réussie parmi les n stations en un slot de temps donné. Une transmission est considérée comme réussie si exactement une station émet pendant cette période de temps. Si p_c est la probabilité qu'il y ait au moins un paquet transmis sur le médium parmi les n stations (p_c est aussi donné dans [83]), nous avons :

$$P_{suc} = (n-1)((1-p_c)^{(n-2)/(n-1)} + p_c - 1)$$

Nous montrons maintenant que le temps moyen au bout duquel tous les voisins d'un nœud ont émis avec succès, est borné par une constante. Soit X la variable aléatoire désignant le nombre de slots de temps nécessaire pour que n stations arrivent à émettre avec succès. Dans le meilleur cas, chaque station parle à tour de rôle. Ceci donne : $\mathbb{P}\left[X < n\right] = 0$ et $\mathbb{P}\left[X = n\right] = P^n_{suc}$.

$\mathbb{P}\left[X = k, k > n\right]$ est la probabilité qu'à la fin des $(k-1)$ premiers slots, $(n-1)$ stations ont émis avec succès et que la n^{ieme} station réussit à transmettre sa trame durant le slot k. Nous avons :

$$\mathbb{P}[X = k, k > n] = \binom{k-1}{k-n+1}\binom{n}{n-1}(1-P_{suc})^{(k-n+1)}P_{suc}^{n}$$

Les $n-1$ premières stations ont émis pendant les $k-1$ premiers slots. On considère donc la probabilité de choisir $k-(n-1)$ slots parmi les $k-1$ premiers slots pendant lesquels aucune transmission n'a eu lieu ou une collision est apparue, multiplié par le nombre de possibilités de choisir la n^{ieme} station qui émet durant le k^{ieme} slot.

On en déduit le nombre moyen de slots nécessaire $\mathbb{E}[X]$ pour que chacune des n stations parvienne à émettre avec succès.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{k=0}^{\infty} k\mathbb{P}[X = k] \\
&= n\mathbb{P}[X = n] + \sum_{k=n+1}^{\infty} k\mathbb{P}[X = k] \\
&= P_{suc}^{n} \times \left(n + \sum_{k=n+1}^{\infty} k\binom{k-1}{k-n+1}\binom{n}{n-1}(1-P_{suc})^{(k-n+1)} \right)
\end{aligned}$$

Ceci peut être dérivé en :

$$\begin{aligned}
\mathbb{E}[X] &= P_{suc}^{n}\left(n + n(n+1)(\frac{1}{P_{suc}^{n}} - (n+1) + nP_{suc}) \right) \\
&= nP_{suc}^{n}\left(1 + (n+1)(\frac{1}{P_{suc}^{n}} - (n+1) + nP_{suc}) \right)
\end{aligned}$$

Ainsi, comme P_{suc} dépend uniquement de n et que nous supposons n borné par une constante, $\mathbb{E}[X]$ est aussi constant. D'où notre hypothèse stipulant "une constante $\tau > 0$ telle que chaque nœud est en mesure de diffuser localement une trame et d'en recevoir une de chacun de ses voisins en un temps borné $\Delta(\tau)$".

Chapitre 4

Diffusion

4.1 Introduction

Comme nous avons pu le constater, auto-organiser un réseau sans fil tel un réseau *ad hoc* ou de capteurs, présente de nombreux avantages. Cependant, de tels réseaux nécessitent également un mécanisme efficace de diffusion d'information. La diffusion (ou *broadcast*) consiste à transmettre un message depuis un nœud source vers l'ensemble des entités du réseau. Une telle opération est employée par la grande majorité des protocoles de routage (pour la découverte des routes entre les entités du réseau). Cette opération s'avère aussi utile à une station de base dans un réseau de capteurs lors de la diffusion d'une requête ou de mise à jour logicielle sur tous les capteurs. Cette opération, indispensable donc à tout réseau sans fil, a fait l'objet de nombreux travaux avec, comme but premier, la réduction du nombre de nœuds retransmettant le message lors de sa diffusion à l'ensemble du réseau.

Les bonnes propriétés d'un protocole de diffusion efficace sont les suivantes :
- extensibilité : il supporte le passage à l'échelle ;
- accessibilité : une grande majorité des nœuds du réseau joignables par la source (appartenant à la même composante connexe) reçoit le message (plus de 90%) ;
- économe : l'énergie et la bande passante consommées sont minimisées (le nombre de messages retransmis et de réceptions redondantes est réduit).

Étant donné qu'un réseau sans fil nécessite à la fois une auto-organisation et un protocole de diffusion, nous proposons d'utiliser la structure d'arbres formée par l'algorithme 1, non seulement pour organiser le réseau en clusters, mais également pour établir une base propice à une diffusion efficace, tirant avantage de certaines de ses caractéristiques. Ainsi, une seule structure est créée pour deux opérations : l'organisation et la diffusion. Notre algorithme de diffusion n'autorise que les nœuds internes des arbres à retransmettre le message. Comme nous l'avons constaté dans le chapitre 3, une grande proportion des nœuds sont des feuilles (environ 75%). Par conséquent, une diffusion basée sur un tel ensemble n'autorise que peu de nœuds à émettre. L'ensemble

des arbres de *clustering* forme une forêt couvrante, donc un ensemble où tout nœud est soit un nœud interne, soit directement voisin d'un nœud interne. Néanmoins, cet ensemble n'est pas connecté puisque les arbres sont indépendants. Pour que la diffusion touche toutes les entités du réseau, il faut tout d'abord connecter ces arbres en établissant des passerelles entre eux. Lors de la diffusion, seuls les nœuds internes et ceux constituant les passerelles seront autorisés à retransmettre le message. Notre algorithme permet deux types de diffusion : une diffusion générale d'un message à tous les nœuds du réseau mais également une diffusion d'un message limitée à l'intérieur d'un cluster. Pour ce dernier cas de figure, comme la hauteur des arbres est petite et proche de l'optimal (excentricité du chef de cluster) et que les clusters sont proches de cellules de Voronoï (chapitre 3), un nœud recevra rapidement une information provenant de son chef.

Dans ce chapitre, nous décrivons dans un premier temps notre algorithme de diffusion reposant sur la structure d'arbres ainsi que les algorithmes de sélection des passerelles. Nous donnons ensuite une analyse théorique d'une diffusion dans tout le réseau, montrant que le nombre de réceptions par nœud peut s'exprimer comme le produit des degrés des relais par la probabilité pour un nœud d'être un relais. Les simulations viennent illustrer notre analyse théorique, comparer sur divers aspects plusieurs protocoles existant et en évaluer la robustesse. Étonnamment, il apparaît que les protocoles les plus fiables ne sont pas ceux produisant le plus de relais mais ceux dont les relais ont le plus fort degré. Les comparaisons entre ces différents algorithmes montrent aussi que notre heuristique de diffusion présente le meilleur compromis entre la consommation d'énergie (nombre d'émissions et réceptions) et la robustesse. Au cours de ces simulations, nous avons pu remarquer que l'heuristique de diffusion basée sur les MPR (multi-points relais) de OLSR (voir section 4.2) présentait très peu de robustesse face à la mobilité des nœuds. Afin de mieux comprendre le comportement des MPR, nous avons analysé cette heuristique plus en détail.

La section 4.2 présente quelques-unes des solutions de diffusion existant pour les réseaux sans fil. La section 4.3 donne l'analyse théorique de la diffusion, utilisant la géométrie stochastique et la distribution de Palm. La section 4.4 présente la façon dont nous utilisons la structure d'arbres sous-jacente afin de réaliser une diffusion efficace. Nos comparaisons et évaluations des différents algorithmes sont menées au travers des simulations de la section 4.5. La section 4.6 présente l'analyse de la sélection des MPR dans OLSR. Enfin, quelques remarques concluront ce chapitre (section 4.7).

4.2 Les algorithmes de diffusion pour les réseaux *ad hoc* dans la littérature

Afin de pouvoir supporter une extension du réseau, un protocole de diffusion dans les réseaux sans fil se doit de limiter l'utilisation de la bande passante et la dépense en énergie ; il doit donc minimiser le nombre de messages générés tout en assurant qu'un maximum de nœuds connectés à la source reçoivent le message (plus de 90%). Beaucoup de solutions ont été proposées avec des hypothèses plus ou moins similaires

à notre modèle. Dans cette section, nous ne mentionnerons que ceux supposant les mêmes hypothèses que nous, c'est à dire qui supposent un modèle à antennes omni-directionnelles, sans contrôle de puissance. De plus, en supposant une couche MAC idéale (qui ne génère aucune collision), on considère la diffusion efficace si tous les nœuds connectés à la source reçoivent le message. Un état de l'art plus complet concernant des solutions probabilistes, utilisant des antennes directionnelles ou considérant une couche MAC non idéale est donné dans [18].

La méthode de diffusion la plus triviale pour diffuser un message est l'inondation aveugle ou *blind flooding* : lorsqu'un nœud reçoit le message diffusé pour la première fois, il le ré-émet pour ses voisins. Ce mécanisme impose une charge énorme au réseau, engendrant un grand nombre de messages et de collisions, dépensant beaucoup d'énergie et de bande passante. C'est pourquoi un tel mécanisme ne peut être envisagé pour un réseau dense ou étendu. Ceci donna motive la mise au point de protocoles de diffusion plus intelligents qui minimisent le nombre de re-transmissions nécessaires en n'autorisant qu'un sous-ensemble de nœuds à transmettre. Pour cela, on cherche à trouver un ensemble "dominant". En effet, afin que tous les nœuds du réseau reçoivent le message, chacun des nœuds doit être soit un dominant, soit voisin d'un dominant. La difficulté est alors de trouver un tel ensemble dominant connexe de taille minimum qui minimise également le nombre de réceptions redondantes d'un message retransmis par cet ensemble. Ce problème est montré NP-difficile [34]. I. Stojmenovic et J. Wu [75] ont proposé une classification des protocoles de diffusion en fonction du type d'ensemble dominant qu'ils utilisent : *cluster-based* ou basé sur la formation de clusters, ensemble dominant dépendant de la source et ensemble dominant indépendant de la source.

Les solutions *cluster-based* [28, 36] sont les plus anciennes. Ces protocoles sont plus détaillés dans le Chapitre 2. L'idée est que chaque nœud ayant le plus petit identifiant (protocole Linked Cluster Architecture - LCA) ou le plus fort degré (High Connectivity Clustering - HCC) dans son 1-voisinage se déclare tête de cluster. Ses voisins s'attachent à lui. Si un nœud s'attache à plus d'un cluster-head, il devient une passerelle. L'ensemble dominant connexe résultant comprend les cluster-heads et les passerelles. Par la suite, des optimisations ont été proposées afin de minimiser la maintenance pour éviter des réactions en chaîne étendues à tout le réseau lors de mouvements de nœuds [24] ou afin de limiter le nombre de passerelles et donc la taille de l'ensemble dominant [81].

Dans les propositions basées sur des ensembles dominants dépendant de la source [49, 65], les émetteurs sélectionnent parmi leurs voisins les nœuds qui relaieront le message. L'ensemble des relais ainsi choisis par un nœud u est aussi petit que possible et tel que, chaque nœud à 2 sauts de u est voisin d'au moins un de ces relais. Pour établir cette sélection, u nécessite une connaissance de son 2-voisinage uniquement. Lors de la diffusion, u fera suivre le message diffusé qu'il reçoit de v, seulement s'il le reçoit pour la première fois et a été choisi comme relais de v. Les différents algorithmes diffèrent ensuite sur la sélection des relais, le plus connu étant celui basé sur les Multi-Points Relais (MPR) de OLSR [65]. Dans OLSR, les MPR sont également utilisés pour établir les tables de routage. La structure de diffusion a donc un usage double. Nous détaillons plus la sélection des MPR dans la section 4.6.

Les protocoles de diffusion utilisant des ensembles dominants indépendants de la source sélectionnent cet ensemble indépendamment du nœud initiateur de la diffusion. C'est le cas de notre algorithme. Les nœuds décident d'eux-mêmes s'ils sont ou non dans cet ensemble, contrairement aux solutions basées sur des ensembles dominants dépendant de la source, ou la décision est prise par un autre nœud. Beaucoup de solutions de ce type ont été proposées. Dans chacune d'elles, les nœuds ne nécessitent que la connaissance de leur 2-voisinage pour prendre leur décision. Un protocole simple et efficace est le NES (*Neighbor Elimination-Based Scheme*) de Wu et Li [80], qui se base sur l'élimination de voisins. Dans ce schéma, un nœud u est dit *intermédiaire* si au moins deux de ses voisins v et w ne sont pas eux-mêmes voisins (u est l'intermédiaire entre v et w). À partir de là, deux règles de sélection sont appliquées sur les intermédiaires afin de réduire leur nombre. Les nœuds restants deviennent les membres de l'ensemble dominant et donc les relais lors d'une diffusion. Les règles de sélection se basent sur une *valeur de priorité*. Dans la version originale du NES, cette valeur est l'identifiant des nœuds. Puis, plusieurs variantes ont été proposées utilisant pour cette valeur le degré du nœud ou l'énergie restante [26, 74]. Les auteurs de [74] proposent un autre type d'algorithme basé sur l'élimination des voisins que l'on peut résumer par *"Wait and See"*. Sur réception d'un message de diffusion, un nœud attend pendant un temps aléatoire. Durant cette période, il observe si un de ces voisins retransmet le message et dans ce cas, quels sont ses voisins recevant ainsi l'information. Si à la fin de la période d'attente, il reste parmi ses voisins des nœuds n'ayant pas reçu le message, il l'émet. Les auteurs de [19] ont ensuite proposé une amélioration à cet algorithme en considérant le RNG - *Relative Neighborhood Graph* (graphe de voisinage relatif) plutôt que le graphe réel. Ces derniers schémas basés sur l'élimination de voisins (*Wait & See* et *Wait & See* basé sur RNG) obtiennent d'excellentes performances en terme de nombre d'émissions et de réceptions mais induisent une latence importante dans le processus de diffusion du fait de la période d'attente aléatoire de chaque nœud.

4.3 Analyse théorique d'une diffusion dans un réseau sans fil

Comme nous avons pu le constater dans la section 4.2, la plupart des protocoles de diffusion visent à réduire le nombre de nœuds qui relaient le message, l'objectif principal étant de minimiser l'énergie globale consommée pour diffuser le message. Comme les nœuds consomment de l'énergie non seulement pour transmettre mais aussi pour recevoir un message, un protocole de diffusion efficace en terme d'économie d'énergie doit chercher à minimiser E, avec $E = C_{tx} \times nb_{tx} + C_{ry} \times nb_{ry}$ où C_{tx} (resp. C_{ry}) est le coût énergétique d'une transmission (resp. réception) d'un paquet et nb_{tx} (resp. nb_{ry}) est le nombre de fois où le message est émis (resp. le nombre de réceptions).

Or, les nœuds des réseaux *ad hoc* utilisant la technologie 802.11 [31], tout comme les capteurs [63], nécessitent approximativement autant d'énergie pour recevoir que pour émettre ($C_{ry} \approx C_{tx}$) et donc, ni les réceptions ni les transmissions ne peuvent être négligées lors du bilan énergétique.

Dans cette analyse, nous nous sommes intéressés au nombre moyen de fois où un nœud donné reçoit un même message lors d'une diffusion. Comme dans les analyses théoriques précédentes, nous utilisons les propriétés des processus ponctuels et les mêmes notations, à savoir : $\Phi(S)$ représente le nombre de points du processus Φ distribués sur la surface S, $B(x,R)$ est la boule de rayon R centrée en x et $B_x^{'} = B(x,R) \setminus \{x\}$.

Nous notons $\overline{r}$ le nombre moyen de réceptions d'un même message par un nœud (qu'il soit un relais ou non). Nous donnons deux résultats pour $\overline{r}$ dans les Propositions 1 et 2, que nous utiliserons par la suite afin de comparer les différents algorithmes de diffusion étudiés. Les résultats de $\overline{r}$ donnés par ces deux propositions sont semblables mais la Proposition 1 considère le modèle particulier que nous utilisons dans les simulations, décrit dans le Chapitre 1.1 alors que les résultats de la proposition 2 sont également applicables à une classe plus large de graphes aléatoires. Dans les deux cas, nous ne donnons ici que l'idée générale de la preuve, les preuves et calculs détaillés se trouvant en Annexes 4.9.

Pour la Proposition 1, nous considérons un processus ponctuel stationnaire Φ d'intensité $\lambda > 0$. Deux points (x,y) de Φ sont connectés (et donc voisins) si et seulement si la distance Euclidienne les séparant est inférieure ou égale à R ($d(x,y) \leq R$), R étant le rayon de transmission radio des nœuds (modèle de graphe géométrique aléatoire).

Proposition 1 *Étant donné un processus ponctuel stationnaire Φ d'intensité λ ($\lambda > 0$), soit Φ_{Relay} d'intensité λ_{Relay} un amincissement de Φ. Les points de Φ_{Relay} représentent les relais. Nous supposons que Φ_{Relay} est toujours un processus ponctuel stationnaire. Le nombre moyen de réceptions d'un même message par nœud $\overline{r}$ est :*

$$\overline{r} = \frac{\lambda_{Relay}}{\lambda} \mathbb{E}^{o}_{\Phi_{Relay}} \left[\Phi(B_0^{'}) \right]$$

où $\mathbb{E}^{o}_{\Phi} \left[\Phi_{Relay}(B_0^{'}) \right]$ est l'espérance sous Palm par rapport au processus Φ (et donc la valeur moyenne) du nombre de relais dans $B_0^{'}$.

La preuve de cette proposition est donnée en annexes 4.9. Le nombre de réceptions d'un même message reçu par un nœud dépend du nombre de relais dans son voisinage. Le résultat s'interprète de la façon suivante. Le nombre moyen de réceptions par nœud est le produit du degré moyen d'un relais par la probabilité pour un nœud donné d'être un relais (ou de façon équivalente par le ratio du nombre de relais sur le nombre total de nœuds).

Nous considérons maintenant des modèles de graphes aléatoires plus généraux. Nous supposons que les degrés des nœuds et le nombre de réceptions par nœud sont équidistribuées. A noter que nous ne supposons pas ces quantités indépendamment distribuées, ce qui fait que cette hypothèse n'est pas restrictive. De plus, cette condition est vérifiée par la plupart des graphes aléatoires. Par exemple, un graphe aléatoire de type Erdös et Renyi [29] qui consiste en n sommets entre lesquels des arêtes sont

placées avec une probabilité uniforme p, indépendamment des autres arêtes, vérifie nos hypothèses.

Proposition 2 *Étant donné un graphe aléatoire $G(V, E)$ et un ensemble de relais $Relay \subset V$ où les degrés des nœuds et des relais ainsi que le nombre de réceptions par nœud sont équi-distribués. Le nombre moyen de réceptions par nœud $\overline{r}$ s'écrit :*

$$\overline{r} \quad = \quad \mathbb{E}\left[\delta(v_1)\middle| v_1 \in Relay\right]\mathbb{P}(v_1 \in Relay) \tag{4.1}$$

La preuve de cette proposition est donnée en annexes 4.9. L'idée est de voir que le nombre de réceptions d'un même message reçues par un nœud correspond au nombre moyen de relais qu'il a dans son voisinage. On peut déduire le résultat ci-dessus, pour un graphe général (proposition 2). Il est le même que pour un graphe géométrique aléatoire (proposition 1) : le nombre moyen de réceptions par nœud est le produit du degré des relais par la probabilité d'être un relais. Dans l'égalité 4.1, v_1 est un nœud choisi aléatoirement parmi l'ensemble des sommets V. Le choix de v_1 n'a aucun impact sur les résultats puisque la probabilité pour un nœud d'être un relais lors de la diffusion est équi-distribuée et est la même pour v_1 que pour tout autre nœud du graphe.

Dans cette analyse, nous avons montré que le nombre moyen de réceptions par nœud est le produit du degré moyen des relais par la proportion des relais. Si n est le nombre de nœuds dans le réseau et prop_{tx} la proportion des relais ($nb_{tx} = n \times \text{prop}_{tx}$), l'énergie globale consommée s'écrit :

$$E = nC_{ry} \times \text{prop}_{tx} \times \left(p + \overline{\delta}_{Relay}\right)$$

où p dépend du type de technologie utilisée par les nœuds radio ($p \approx 1$ pour les capteurs [63], $p \approx 4$ pour les nœuds utilisant la technologie 802.11 [31]). Il en ressort clairement que pour diminuer E, il faut jouer sur les paramètres $\overline{\delta}_{Relay}$ et prop_{tx}.

4.4 Notre contribution à la diffusion

Dans cette section, nous introduisons dans un premier temps un algorithme permettant l'élection de nœuds passerelles entre nos clusters. Puis, dans un second temps, nous donnons l'algorithme permettant d'appliquer deux types de diffusion sur notre organisation en arbres : un algorithme de diffusion globale (dans tout le réseau) et un algorithme de diffusion dans un cluster.

4.4.1 Sélection des passerelles

On appelle nœud frontière un nœud comptant parmi ses voisins au moins un représentant d'un ou plusieurs clusters autres que le sien.

Une passerelle $\mathcal{G}ateway(\mathcal{C}(u), \mathcal{C}(v)) = \langle x, y\rangle$ entre deux clusters voisins $\mathcal{C}(u)$ et $\mathcal{C}(v)$ est une paire de nœuds frontières $\langle x, y\rangle$ telle que $x \in \mathcal{C}(u)$, $y \in \mathcal{C}(v)$ et $x \in \Gamma_1(y)$.

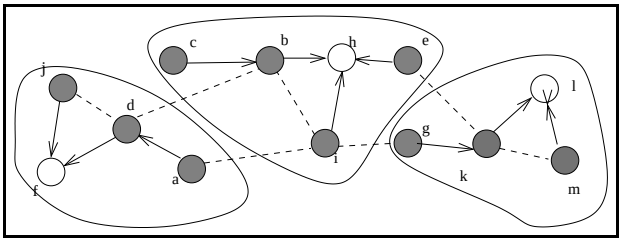


FIG. 4.1 – Exemple d'arbres de *clustering* formés par la métrique de densité.

Dans une telle paire, on appelle le nœud x le *nœud passerelle* $x = GW(\mathcal{C}(u), \mathcal{C}(v))$ et le nœud y le nœud *miroir* de la passerelle $y = GWm(\mathcal{C}(u), \mathcal{C}(v))$. Ces passerelles sont *orientées* dans le sens où il existe une passerelle permettant à $\mathcal{C}(u)$ de joindre $\mathcal{C}(v)$ ($\mathcal{G}ateway(\mathcal{C}(u), \mathcal{C}(v))$) et une autre qui permet à $\mathcal{C}(v)$ de joindre $\mathcal{C}(u)$ ($\mathcal{G}ateway(\mathcal{C}(v), \mathcal{C}(u))$), ces deux passerelles pouvant être différentes.

Notre algorithme de sélection des passerelles se déroule en deux étapes. Dans un premier temps, chaque nœud frontière, choisit localement son miroir dans les clusters voisins. Un nœud frontière et son miroir forment alors une paire de nœuds candidate au titre de passerelle. Dans un second temps, l'algorithme sélectionne parmi ces paires candidates, la paire la plus adéquate au rôle de passerelle. Comme les nœuds de la passerelle seront invités à re-transmettre un message diffusé dans tout le réseau, l'algorithme de sélection favorise l'élection des nœuds internes afin de minimiser le nombre d'émetteurs. En effet, les nœuds internes appartiennent déjà à l'ensemble des nœuds relais. Les sélectionner en tant que passerelle n'ajoute aucun émetteur et donc aucun message superflu.

Cependant, il est clair que si une passerelle est le seul moyen de raccorder un ensemble du réseau à la source du message diffusé, cette passerelle devient un point sensible. Afin d'ajouter de la robustesse au protocole envers une cassure de liens au niveau des passerelles, chaque nœud passerelle élit parmi ses voisins une passerelle de secours (ou de *back-up)*. Cette dernière ré-émettra le message diffusé si et seulement si elle n'entend pas la passerelle principale. Nous reprenons ici la philosophie *Wait & See* vue dans la section 4.2. Dans la suite, nous détaillons la sélection des trois types de nœuds : miroir, passerelle et passerelle de secours.

Sélection des miroirs.

Comme mentionné dans le chapitre 3.8, tout nœud u sait en un temps borné s'il existe parmi ses voisins un nœud v qui n'appartient pas au même cluster que lui ($\mathcal{C}(u) \neq \mathcal{C}(v)$) et donc s'il est un nœud frontière. Chaque nœud frontière u doit sélectionner son *miroir* parmi les nœuds de son voisinage appartenant à un cluster différent du sien. Pour cela, dans un premier temps, u considère parmi ces nœuds ceux qui ne sont pas des feuilles et qui sont donc des transmetteurs dans tous les cas. u sélectionne parmi eux le nœud de plus forte densité. Si tous les nœuds considérés sont des feuilles, u choisit

le nœud de plus faible degré, de façon à limiter le nombre de réceptions occasionnées lors de l'émission du message diffusé par le miroir.

Si u est un nœud frontière du cluster $\mathcal{C}(v)$ ($\mathcal{C}(v) \neq \mathcal{C}(u)$), on note $m(u, \mathcal{C}(v))$ le nœud miroir choisi par u dans $\mathcal{C}(v)$. Si u est voisin de plusieurs clusters différents du sien, il doit élire plusieurs miroirs, un dans chacun des clusters voisins. Par exemple, sur la figure 4.1, le nœud i doit se choisir deux miroirs, $m(i, \mathcal{C}(f))$ dans $\mathcal{C}(f)$ et $m(i, \mathcal{C}(l))$ dans $\mathcal{C}(l)$.

Algorithm 1 Sélection du nœud miroir - EXÉCUTÉ SUR CHAQUE NŒUD FRONTIÈRE u, *c.a.d.,* $\exists v \in \Gamma_1(u) \,\text{s.t.}\, \mathcal{C}(v) \neq \mathcal{C}(u)$

Pour chaque cluster voisin $\mathcal{C}$ pour lequel u est un nœud frontière : $\mathcal{C} \neq \mathcal{C}(u)$ et $\exists v \in \Gamma_1(u) \cap \mathcal{C}$

Sélectionne l'ensemble S des nœuds tels que $S = \mathcal{C} \cap \{v \mid \Gamma_1(u) \,|\, \mathcal{C}h(v) \neq \emptyset\}$.

▷ *u considère dans un 1^{er} temps l'ensemble des nœuds non feuilles, émetteurs dans tous les cas.*

if ($S \neq \emptyset$) **then** Sélectionne l'ensemble S' des nœuds tels que $S' = \{v \mid v = max_{w \in S} \rho(w)\}$.

▷ *u considère les candidats ayant la plus forte valeur de densité dans le but de favoriser la stabilité.*

else ▷ *Tous les candidats miroirs de u sont des feuilles.*

$S = \{\mathcal{C} \cap \Gamma_1(u)\}$.

Sélectionne l'ensemble S' des nœuds tels que $S' = \{v \,|\, v = min_{w \in S} \delta(w)\}$.

▷ *u considère les feuilles de plus faible degré afin de minimiser le nombre de réceptions générées lors de l'ajout de cette feuille dans l'ensemble des relais.*

end

if ($S' = \{v\}$) **then** $m(u, \mathcal{C}) = v$.

▷ *S' ne contient qu'un nœud : le miroir de u.*

else $m(u, \mathcal{C}) = v$ tel que $Id(v) = min_{w \in S'} Id(w)$.

▷ *Il existe des ex-aequo. u choisit le nœud de plus faible Id.*

end

Sélection des passerelles.

La seconde étape de l'algorithme de sélection de passerelle élit les passerelles reliant chaque cluster $\mathcal{C}$ à chacun de ses clusters voisins $\mathcal{C}'$, parmi les paires formées par un nœud frontière et son miroir dans $\mathcal{C}'$. Cette étape nécessite que des informations concernant les nœuds frontières soient remontées à la racine. Suivant la taxonomie de [81], cette étape est qualifiée de quasi-locale car chaque entité nécessite des informations situées à une distance bornée (ici distance bornée par la hauteur de l'arbre, elle-même bornée par une constante). La première étape de l'algorithme qui permet aux nœuds frontières de sélectionner leur miroir ne nécessite que des informations locales (de voisinage) et est qualifiée de locale. Le fait que ces étapes soient locales ou quasi-locales sous-entend une maintenance rapide et une robustesse de l'algorithme envers la mobilité des nœuds [81].

La sélection des passerelles de notre algorithme est distribuée puisqu'une sélection est opérée à chaque niveau de l'arbre. Tout comme la sélection des miroirs, elle cherche à favoriser l'élection des nœuds internes en tant que passerelles de façon à limiter les réceptions redondantes lors d'une diffusion d'un message. Les nœuds frontières envoient leur Id à leur père en leur indiquant si eux-mêmes et leur miroirs sont ou non des feuilles. Chaque père choisit parmi tous ses fils frontières le meilleur candidat dont il envoie les informations à son propre père et ainsi de suite, jusqu'à atteindre le *cluster-head*. La sélection est donc semi-distribuée puisque chaque nœud interne élimine des candidats et n'en renvoie qu'un seul à son père. De cette façon, seuls des paquets de petites tailles sont envoyés depuis les nœuds frontières jusqu'à la tête de cluster. Comme mentionné en section 3.6.2-table 3.5, le degré moyen des nœuds internes est faible et constant quel que soit le nombre de nœuds, ce qui induit un nombre borné de messages à chaque niveau. De plus, comme la hauteur des arbres est également bornée par une constante, le nombre de niveaux est lui aussi faible.

Définition 2 (Sous-arbre) *v appartient au sous-arbre de racine u (noté $s\mathcal{T}(u)$) si l'une des trois conditions suivantes est remplie :*
– *$u = v$,*
– *u est le père de v : $u = \mathcal{P}(v)$,*
– *le père de v appartient au sous-arbre de racine u : $\mathcal{P}(v) \in s\mathcal{T}(u)$.*

$\mathcal{C}(x)$ est un cluster voisin du sous-arbre $s\mathcal{T}(u)$ si et seulement si $\mathcal{C}(x) \neq \mathcal{C}(u)$ et il existe dans $s\mathcal{T}(u)$ un nœud v frontière du cluster $\mathcal{C}(x)$: $\exists z \in \mathcal{C}(x)$ et $y \in s\mathcal{T}(u)$ tels que $y \in \Gamma_1(z)$.

La passerelle entre deux clusters voisins est alors sélectionnée de la façon suivante. L'algorithme est exécuté par chaque nœud interne, après réception des informations concernant tous les nœuds frontières de son sous-arbre. Pour chaque cluster voisin de son sous-arbre, un nœud interne u considère l'ensemble G des nœuds candidats (nœuds frontières) ($G = \{v \in s\mathcal{T}(u) \mid \exists w \in \Gamma_1(v) \mid \mathcal{C}(w) \neq \mathcal{C}(u)\}$). Il sélectionne parmi eux le sous-ensemble $G' \subset G$ des nœuds internes. Si G est seulement composé de nœuds feuilles (et donc $G' = \emptyset$), la sélection se poursuit parmi les nœuds de G directement. Le nœud u prend en priorité les nœuds dont le miroir est un nœud interne et il choisit parmi eux le nœud de plus forte densité si les candidats sont des nœuds internes ou de plus faible degré sinon. En cas d'égalité, le nœud de plus petit identifiant est élu. On remarque qu'entre deux clusters voisins $\mathcal{C}(u)$ et $\mathcal{C}(v)$, il existe deux passerelles $\mathcal{G}ateway(\mathcal{C}(u), \mathcal{C}(v))$ et $\mathcal{G}ateway(\mathcal{C}(v), \mathcal{C}(u))$ qui sont différentes dans la plupart des cas. Du fait de leur orientation et comme un relais ne re-transmet que sur la première réception du message, dans la plupart des cas, seulement une de ces deux passerelles sera utilisée lors d'une diffusion. Ce phénomène sera mis en évidence par les simulations de la section 4.5.

Algorithm 2 Sélection des passerelles - EXÉCUTÉ PAR CHAQUE NŒUD INTERNE u

Pour chaque cluster $\mathcal{C} \neq \mathcal{C}(u)$ **pour lequel** $\exists v \in s\mathcal{T}(u)$ **nœud frontière**
Considère l'ensemble G des candidats : $G = \{v \in s\mathcal{T}(u) \mid \exists w \in \Gamma_1(v) \mid \mathcal{C}(w) = \mathcal{C}\}$.
Sélectionne l'ensemble $G' \subset G$ des nœuds v tel que $G' = G \cap \{v|\mathcal{C}h(v) \neq \emptyset\}$.
▷ *u considère en priorité les nœuds non feuilles.*
if ($G' \neq \emptyset$) **then**
▷ *u favorise les nœuds internes de plus forte densité ayant un nœud non feuille comme miroir.*

Sélectionne l'ensemble $G" \subset G'$ tel que $G" = G' \cap \{v|\mathcal{C}h(m(v,\mathcal{C})) \neq \emptyset\}$.
if ($G" \neq \emptyset$) **then**
Sélectionne l'ensemble $Finalist \subset G"$ tel que
$Finalist = \{v|\rho(v) = max_{w \in G"}\rho(w)\}$.
▷ *Passerelle Nœud Interne↔Nœud Interne.*
else
Sélectionne l'ensemble $Finalist \subset G"$ tel que
$Finalist = \{v|\rho(v) = max_{w \in G'}\rho(w)\}$.
▷ *Passerelle Nœud Interne↔Feuille.*
end
else
▷ *Tous les candidats sont des feuilles. u favorise ceux de plus faible degré ayant un nœud interne comme miroir.*
Sélectionne l'ensemble $G" \subset G$ tel que $G" = G \cap \{v|\mathcal{C}h(m(v,\mathcal{C})) \neq \emptyset\}$.
if ($G" \neq \emptyset$) **then**
Sélectionne $Finalist \subset G"$ tel que $Finalist = \{v|\delta(v) = min_{w \in G"}\delta(w)\}$.
▷ *Passerelle Feuille↔Nœud Interne.*
else
Sélectionne $Finalist \subset G"$ tel que $Finalist = \{v|\delta(v) = min_{w \in G'}\delta(w)\}$.
▷ *Passerelle Feuille↔Feuille.*
end
end
if ($Finalist = \{v\}$) **then**
$Winner = v$.
else
$Winner = \{v|Id(v) = min_{w \in Finalist}Id(w)\}$.
▷ *Conflits. u choisit le nœud de plus petit Id.*
end
if ($u = \mathcal{H}(u)$) **then**
$Winner$ est le nœud passerelle :
$\mathcal{G}ateway(\mathcal{C}(u), \mathcal{C}) = \langle Winner, m(Winner, \mathcal{C})\rangle$.
else
Envoie l'identité de $Winner$ à son père $\mathcal{P}(u)$.
end

Sélection de la passerelle de secours.

Cette sélection est purement locale et n'engendre aucun coût supplémentaire. Elle tire avantage du caractère de diffusion du médium radio qui fait que lorsqu'un nœud émet, tous les nœuds à portée radio entendent le message, même s'il ne leur est pas destiné. Quand un nœud frontière u envoie une information à son père durant le processus de

sélection des passerelles (algorithme 2), chacun de ses voisins apprend la condition de u (feuille, nœud interne, nœud frontière, etc.). De cette façon, le nœud passerelle apprend qui dans son voisinage était également candidat et ainsi peut servir potentiellement de passerelle de secours. Il sélectionne ce nœud en choisissant parmi ses voisins un nœud frontière dont le miroir est différent du sien. Cette passerelle de secours agit de la façon suivante. Sur réception d'un message de diffusion, la passerelle de secours enclenche un compte à rebours. Si à la fin de celui-ci, elle n'a pas entendu la passerelle principale émettre, elle émet le message. Ce mécanisme n'ajoute aucune réception redondante et ajoute de la robustesse au processus de diffusion.

4.4.2 L'algorithme de diffusion

Dans un réseau sans fil, un nœud peut avoir usage de trois sortes de *broadcast* :
– une diffusion de voisinage : envoi d'un message à tous ses 1-voisins (comme les paquets HELLO) ;
– une diffusion localisée : diffusion dans un cluster uniquement ;
– une diffusion globale : diffusion d'un message dans tout le réseau.
Les passerelles ne seront utilisées que dans le cas d'une diffusion globale, afin de relayer le message diffusé d'un cluster à l'autre. Afin de distinguer ces trois types de diffusions à la réception d'un message, un nœud nécessite une indication dans le paquet reçu[1]. Quand une diffusion est effectuée dans un cluster $\mathcal{C}(u)$, le message est relayé par tous les nœuds internes appartenant à ce cluster. Quand le message doit être propagé dans tout le réseau, tous les nœuds internes du réseau ainsi que les passerelles ré-émettent le message pour leurs voisins. Les passerelles (principales et de secours) étant orientées, elles ne ré-émettent que sous certaines conditions. Le nœud passerelle $GW(\mathcal{C}(u),\mathcal{C}(w))$ re-transmet un message seulement s'il arrive de son propre cluster $\mathcal{C}(u)$. Un nœud passerelle miroir $GWm(\mathcal{C}(u),\mathcal{C}(w))$ ne re-transmet le message que s'il arrive du cluster $\mathcal{C}(u)$ pour lequel il est miroir. Ainsi, un nœud passerelle miroir $GWm(\mathcal{C}(u),\mathcal{C}(w))$ re-transmet un message provenant de $\mathcal{C}(u)$ quel que soit le nœud qui le lui envoie et qui n'est donc pas nécessairement le nœud passerelle $GW(\mathcal{C}(u),\mathcal{C}(w))$. Les passerelles de secours agissent comme décrit dans la section 4.4.1.

Algorithm 3 Algorithme de diffusion

Pour tout nœud u**, sur réception d'un message** M **provenant d'un nœud** $v \in \Gamma_1(u)$
▷ *A noter que* v *est le nœud qui a transmis* M *à* u *mais pas forcément la source de la diffusion.*
  if (u reçoit M pour la première fois) **then**
    if *Diffusion générale* **then**
      if ($\mathcal{C}h(u) \neq \emptyset$) **then**
        Ré-émet
        ▷ u *est un nœud interne.*
      else
        if (($\mathcal{C}(u) = \mathcal{C}(v)$) et ($u = GW(\mathcal{C}(u),\mathcal{C}(w)) \forall w \in V$)) **then**
          ▷ u *est un nœud passerelle et* M *provient de son propre cluster.*

[1] Les adresses IPv6 utilisent déjà ce "scope" d'indication d'adresse multicast : local, global.

```
                Ré-émet
            end
            if ((C(u) ≠ C(v)) et (u = GWm(C(v), C(u)))) then
                ▷ M provient d'un cluster pour lequel u est une passerelle miroir.
                Ré-émet
            end
        end
    else
    ▷ Il s'agit d'une diffusion dans un cluster.
    ▷ M n'est ré-émis que par les nœuds internes dudit cluster.
        if ((C(v) = C(u)) et (Ch(u) ≠ ∅)) then Ré-émet end
    end
end
```

4.5 Analyses et résultats de simulations

Dans un premier temps, nous avons simulé le processus de sélection des passerelles afin de l'évaluer. Puis, nous avons simulé des diffusions globales dans tout le réseau et restreintes à des clusters uniquement, en utilisant notre algorithme ainsi que d'autres protocoles existants afin de comparer les performances de chacun des protocoles de diffusion et de valider les résultats analytiques obtenus dans la section 4.3.

4.5.1 Élection et utilisation des passerelles

Étant donnés deux clusters voisins $\mathcal{C}(u)$ et $\mathcal{C}(v)$, quatre types de passerelles sont possibles :

- Passerelle Feuille↔Feuille : $GW(\mathcal{C}(u), \mathcal{C}(v))$ et $GWm(\mathcal{C}(u), \mathcal{C}(v))$ sont deux nœuds feuilles. Ce type de passerelle est le plus coûteux puisque son utilisation ajoute deux relais dans le processus de diffusion et cause donc plus de réceptions redondantes.
- Passerelle Feuille↔Nœud interne : $GW(\mathcal{C}(u), \mathcal{C}(v))$ est une feuille et $GWm(\mathcal{C}(u), \mathcal{C}(v))$ est un nœud interne. Ce type de passerelle n'ajoute qu'un seul relais. Comme nous le verrons, c'est le type de passerelle le plus élu.
- Passerelle Nœud Interne↔Feuille : $GW(\mathcal{C}(u), \mathcal{C}(v))$ est un nœud interne et $GWm(\mathcal{C}(u), \mathcal{C}(v))$ est un nœud feuille.
- Passerelle Nœud Interne↔Nœud Interne : $GW(\mathcal{C}(u), \mathcal{C}(v))$ et $GWm(\mathcal{C}(u), \mathcal{C}(v))$ sont deux nœuds internes. Ce type de passerelle est le moins coûteux puisqu'il n'ajoute aucun émetteur et donc n'engendre aucune réception superflue. Bien que cela soit le type de passerelle que l'algorithme cherche à favoriser, il est le moins courant.

La table 4.1 donne le nombre moyen de passerelles qu'un cluster doit élire et maintenir en moyenne vers ses clusters voisins, ainsi que le nombre de passerelles qui sont effectivement utilisées lors de la diffusion d'un message dans tout le réseau. On remarque

	500 nœuds	600 nœuds	700 nœuds
#clusters	11.93	11.64	11.36
#passerelles élues par cluster	5.86	6.02	6.16
#passerelles utilisées par cluster	1.76	1.74	1.73
	800 nœuds	900 nœuds	1000 nœuds
#clusters	11.30	11.14	10.72
#passerelles élues par cluster	6.20	6.22	6.26
#passerelles utilisées par cluster	1.76	1.68	1.66

TAB. 4.1 – Nombre de passerelles élues et utilisées par cluster lors d'une diffusion générale initiée par une source choisie aléatoirement.

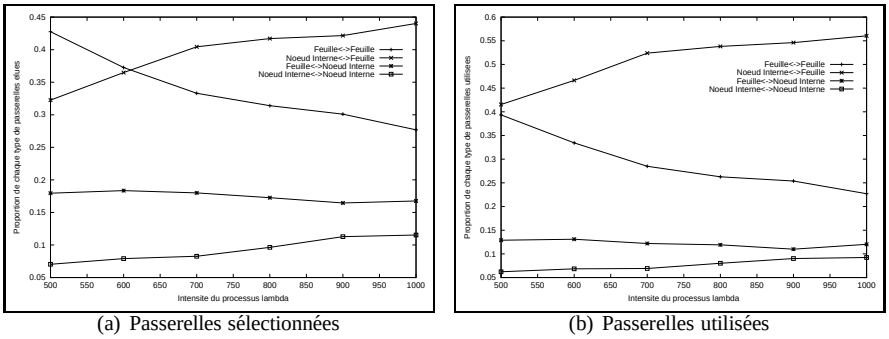
que le nombre de passerelles à élire est raisonnable et quasiment constant malgré l'augmentation de l'intensité des nœuds. Ceci démontre une bonne caractéristique pour l'extensibilité de notre heuristique. Néanmoins, cette propriété était prévisible puisque, comme constaté dans le chapitre 3.6.1, le nombre de clusters est constant à partir d'une certaine intensité du processus sous-jacent et que les clusters formés correspondent grossièrement à des cellules de Voronoï centrées sur les cluster-heads. Comme, dans un diagramme de Voronoï, une cellule a 6 cellules voisines en moyenne, il en est de même pour nos clusters et donc pour le nombre de passerelles qu'ils doivent élire.

La figure 4.2(a) donne la proportion de chaque type de passerelles élues. On remarquera que les deux types de passerelles qu'on retrouve le moins sont celles de type Feuille↔Nœud Interne et Nœud Interne↔Nœud Interne. Cela s'explique par le fait que, par construction, la majeure partie des nœuds frontières sont des feuilles et donc, la majeure partie des miroirs également. Comme l'algorithme de sélection considère en priorité les nœuds internes pour le nœud passerelle, aussitôt qu'un nœud non feuille est candidat, il est sélectionné. Comme il y a une majorité de feuilles sur les frontières, ce nœud interne a une forte probabilité d'avoir une feuille comme miroir. Ceci explique la forte proportion des passerelles Nœud Interne↔Feuille et Feuille↔Feuille. Plus le réseau est dense, plus on a de chances de trouver des nœuds internes aux frontières. C'est pourquoi la proportion de passerelles de type Nœud Interne↔Feuille augmente avec le nombre de nœuds alors que la proportion de passerelles Feuille↔Feuille décroît.

Quand une diffusion générale est effectuée, toutes les passerelles ne sont pas nécessairement utilisées. Si deux clusters voisins $\mathcal{C}(u)$ et $\mathcal{C}(v)$ sont reliés par deux passerelles $\mathcal{G}ateway(\mathcal{C}(u), \mathcal{C}(v))$ et $\mathcal{G}ateway(\mathcal{C}(v), \mathcal{C}(u))$, dans la plupart des cas, seulement une des deux sera utilisée. Comme le montre la table 4.1, le nombre de passerelles utilisées est quasiment constant et reste faible, toujours compris entre 1 et 2. Cela signifie que de façon générale, ou le message diffusé pénètre un cluster et y meurt (dans ce cas, il n'utilise qu'une seule passerelle), ou il le traverse et dans ce cas utilise deux passerelles (une pour entrer et une pour en sortir). Ce phénomène s'explique par le fait que nous considérons une couche MAC idéale et que le message se propage à la même vitesse dans toutes les directions. Un message ne va donc pas "contourner" un cluster

avant de l'inonder.

La figure 4.2(b) illustre la proportion de chaque type de passerelle utilisée lors d'une diffusion globale. La majorité des passerelles utilisées sont celles n'ajoutant qu'un seul relais dans le processus de diffusions. Cela est vrai même pour des petits nombres de nœuds alors que les passerelles Feuille↔Feuille étaient majoritairement élues. Cela montre une nouvelle caractéristique d'extensibilité de notre algorithme de diffusion : il favorise l'utilisation des nœuds internes. Ainsi, comme le nombre moyen de passerelles utilisées est faible et que chacune d'elles n'ajoute qu'un relais dans le processus de diffusion, le coût introduit par ces passerelles est faible.



(a) Passerelles sélectionnées (b) Passerelles utilisées

FIG. 4.2 – Proportion de chaque type de passerelles sélectionnées et utilisées par cluster.(+ : Feuille↔Feuille ; × : Nœud Interne↔Feuille ; ∗ : Feuille↔Nœud Interne ; □ : Nœud Interne↔Nœud Interne)

4.5.2 Performances de la diffusion

De façon à évaluer notre algorithme, nous avons choisi de le comparer à des protocoles de diffusion existants les plus représentatifs (cf. section 4.2) : blind flooding, HCC [36] (schémas cluster-based), Multi-Point Relais (MPR) [65] (ensemble dominant dépendant de la source), le NES de Wu et Li [80] (ensemble dominant indépendant de la source) et le schéma "Wait & See" de I. Stojmenovic, M. Seddigh et J. Zunic [74] (ensemble dominant basé sur des valeurs aléatoires). Comme mentionné lors des analyses théoriques de la section 4.3, nous cherchons à comparer ces algorithmes en terme d'économie d'énergie (nombre de messages envoyés et reçus par nœud) et de bande passante (nombre de messages envoyés au total), en cherchant à calculer l'impact du degré des nœuds relais sur ces performances. C'est pourquoi nous avons relevé la proportion des nœuds qui ré-émettent le message diffusé, le degré de ces nœuds relais ainsi que l'impact de ces valeurs sur le nombre de copies redondantes d'un même message reçues par nœud.

Dans la même optique, nous avons simulé deux variantes du protocole NES de Wu et Li : la version originale [80] où la valeur de priorité utilisée par les nœuds est l'identi-

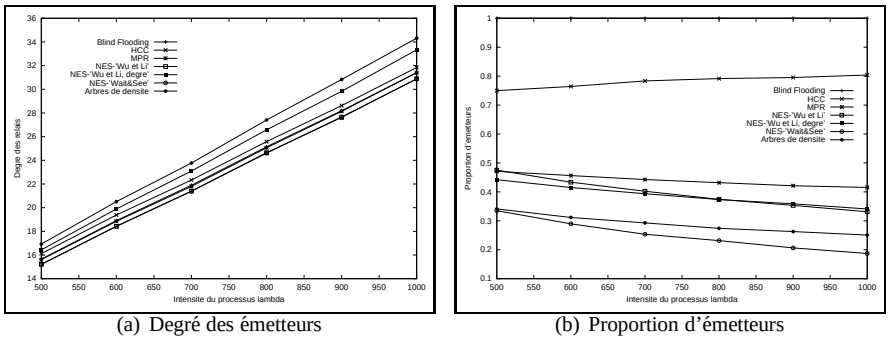
fiant des nœuds, et une version où la valeur de priorité est le degré des nœuds (plus l'Id pour résoudre les conflits) [26].

Nous avons également mesuré la latence[2], excepté pour le protocole NES-"Wait & See" où la latence dépend de la taille de la fenêtre dans laquelle les nœuds tirent un temps d'attente aléatoire.

A priori, un grand nombre de nœuds émetteurs et de réceptions multiples ajoute de la redondance au protocole et le rend théoriquement plus résistant face à la mobilité des nœuds et aux cassures de lien. C'est pourquoi, nous nous sommes intéressés à l'impact du nombre de réceptions redondantes et du degré des relais sur la robustesse des différents protocoles de diffusion.

Diffusion d'un message dans tout le réseau (diffusion générale)

Nous analysons ici une diffusion d'un message dans tout le réseau, initiée par une source choisie aléatoirement parmi les nœuds du réseau.



(a) Degré des émetteurs (b) Proportion d'émetteurs

FIG. 4.3 – Degré *(a)* et proportion *(b)* des émetteurs en fonction des différents algorithmes de diffusion et du nombre de nœuds.(+ : Blind Flooding ; × : HCC ; ∗ : MPR ; □ : NES - Wu Li ; ■ NES - Degré - Wu Li ; ⊖ NES - Wait & See ; ● Arbres de densité)

La figure 4.3 montre le degré dans le graphe des nœuds relais ainsi que leur proportion dans le réseau pour les différents algorithmes de diffusion considérés.

La figure 4.3(a) montre le degré des relais. Comme dans le *Blind Flooding*, tous les nœuds relaient le message, le degré moyen des relais correspond exactement au degré moyen des nœuds dans le graphe. Nous pouvons voir que notre algorithme maximise le degré moyen des relais. En effet, notre algorithme élit les relais sur leur valeur de densité qui est quasiment proportionnelle à leur degré. La version originale du NES de Wu et Li et le NES-"Wait & See" élisent des relais de même degré, inférieur au degré moyen. Ceci est dû au fait que plus un nœud a de voisins, plus il a de chances

[2]Temps au bout duquel tous les nœuds du réseau ont reçu le message de diffusion.

soit d'entendre l'un d'eux émettre avant la fin de son temps d'attente (dans le cas du protocole "Wait & See") soit d'avoir été "éliminé" par les règles de sélection de l'algorithme de Wu et Li. On remarque également que les courbes représentant les degrés des MPR et des relais dans le *Blind Flooding* sont confondues. Ceci montre que les relais utilisés lors d'une diffusion avec les MPR sont choisis indépendamment de leur degré. Les protocoles HCC et le NES-degré Wu de Li choisissent des relais de fort degré par construction, ce qui se retrouve dans les résultats.

La figure 4.3(b) montre la proportion de relais dans le réseau. Comme dans le *blind flooding*, tous les nœuds re-transmettent le message, cette proportion est égale à 1. On observe que le protocole "Wait & See" est l'heuristique nécessitant le moins de relais, ce qui implique que cette heuristique dépense moins d'énergie en émission. Notre heuristique obtient des résultats proches. On remarquera également que les deux variantes du protocole NES de Wu et Li génèrent approximativement le même nombre de relais.

Cependant, comme vu lors des analyses théoriques, le nombre de réceptions par nœud ne peut être directement déduit des résultats du degré des relais ou directement de la proportion d'émetteurs puisqu'il est en fait le produit des deux. Comme certaines des heuristiques produisant le moins de relais sont également celles dont les relais ont les plus forts degrés, nous ne pouvons en déduire laquelle minimise le plus le nombre de réceptions. Nous pouvons juste supposer que la variante du protocole NES utilisant l'identifiant des nœuds induira moins de réceptions que la variante utilisant le degré des nœuds puisque pour un nombre équivalent de ré-émetteurs, le degré de ses relais est plus faible. La figure 4.4 montre le nombre de réceptions par nœud d'un message diffusé dans tout le réseau.

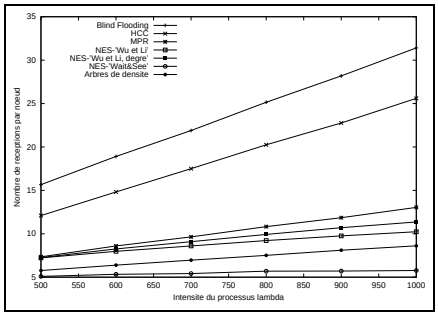
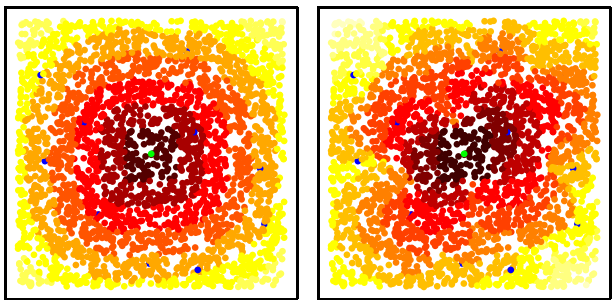


FIG. 4.4 – Nombre de réceptions par nœud en fonction du nombre de nœuds et des différents algorithmes de diffusion.($+$: Blind Flooding ; $\times$: HCC ; $*$: MPR ; $\square$: NES - Wu Li ; $\blacksquare$ NES - Degré Wu Li ; $\ominus$ NES - Wait & See ; $\bullet$ Densité)

Ainsi, lorqu'un message est diffusé à tous les nœuds du réseau, l'algorithme NES-"Wait & See" est celui induisant le moins de réceptions redondantes sur les nœuds, dépensant ainsi moins d'énergie et de ressources. On remarque que notre algorithme obtient des résultats très proches puisqu'il ne génère qu'une réception de plus en moyenne par nœud que "Wait & See". De plus, le "Wait & See" étant basé sur des valeurs aléatoires, la latence qu'il introduit est inévitablement supérieure à celle intro-

duite par notre protocole. On remarque également que la version originale du NES de Wu et Li cause moins de réceptions que sa variante basée sur le degré des nœuds.

Latence : Puisque dans l'algorithme de sélection des MPR, les relais sont choisis de façon à ce que le 2-voisinage d'un nœud soit atteint en 2 sauts, le k-voisinage de la source est atteint en k sauts. Sous l'hypothèse d'une couche MAC idéale, les MPR donnent des résultats optimaux en terme de latence (ici équivalente aux nombres de sauts). C'est pourquoi nous avons comparé la latence produite par notre heuristique à celle produite par les MPR afin de mesurer l'écart entre notre solution et l'optimal. Nous considérons une unité de temps comme une étape de transmission (*c.à.d.* 1 saut). Les résultats sont présentés dans la table 4.2. Même si notre algorithme n'est pas optimal en terme de latence, il ne s'en éloigne guère (2 sauts au plus). La figure 4.5 représente la propagation temporelle d'une diffusion générale d'un message, initiée au temps 0 par une source centrale (en vert sur les schémas). Les *cluster-heads* apparaissent en bleu. La couleur des autres nœuds dépend du temps au bout duquel ils reçoivent le message. Plus la couleur est claire, plus le temps est long.



(a) Propagation avec MPR (b) Propagation avec les arbres de densité

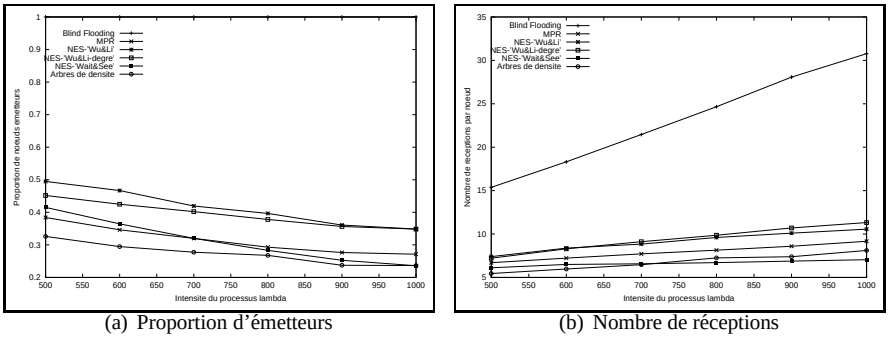
FIG. 4.5 – Temps de propagation d'un message diffusé dans tout le réseau par une source centrale en utilisant les MPR (a) et notre métrique (b).

	500 nœuds		700 nœuds		800 nœuds		900 nœuds		1000 nœuds	
	MOY	MAX	MOY	MAX	MOY	MAX	MOY	MAX	MOY	MAX
MPR	5.13	8.97	4.88	8.40	4.88	8.40	4.81	8.23	4.78	8.07
Densité	6.31	11.05	6.22	10.78	6.24	10.95	6.15	10.66	6.19	10.74

TAB. 4.2 – Temps max et moyen pour recevoir le message. *Les valeurs "MAX" donnent le temps au bout duquel tous les nœuds du réseau ont reçu le message. Les valeurs "MOY" donnent le temps moyen au bout duquel un nœud reçoit le message.*

Diffusion dans un cluster.

On suppose maintenant qu'une diffusion est initiée dans chaque cluster, par chaque cluster-head. Nous avons donc autant de diffusions simultanées que de clusters. Un nœud applique le protocole de diffusion (quel qu'il soit) en ne considérant que les nœuds appartenant au même cluster que lui.



(a) Proportion d'émetteurs (b) Nombre de réceptions

FIG. 4.6 – Proportion de émetteurs (a) et Nombre moyen de réceptions par nœud (b) pour une diffusion dans un cluster en fonction de l'intensité du processus et du protocole utilisé.($+$: Blind Flooding ; $\times$: MPR ; $*$: NES - Wu Li ; $\square$: NES - Degré Wu Li ; $\blacksquare$ NES - Wait & See ; $\ominus$ Densité).

On remarque sur la figure 4.6, que pour ce type de diffusion, notre algorithme est celui qui minimise le plus le nombre de relais et de réceptions, obtenant même des performances égales ou supérieures à celles du protocole NES "Wait & See".

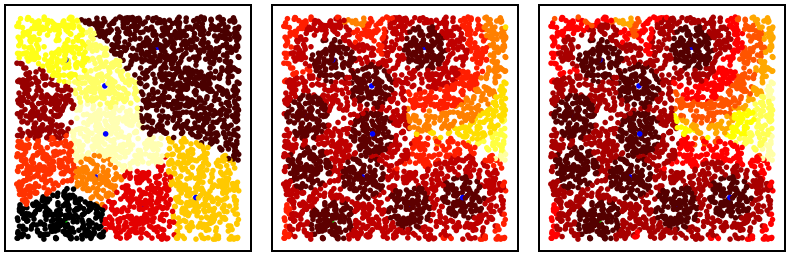
Ces résultats confirment également les résultats analytiques montrant que le nombre de réceptions ne peut être déduit directement de la proportion d'émetteurs. Par exemple, notre algorithme utilise moins de relais avec de plus forts degrés que le "Wait & See" pour finalement induire autant de réceptions par nœud.

La table 4.3 et la figure 4.7 représentent les résultats concernant la latence induite dans de telle diffusion par notre heuristique. Comme pour une diffusion générale, la latence obtenue est très proche de l'optimale, s'en éloignant seulement d'une demie unité de temps pour le temps moyen. Cela montre également que, même si les routes dans les arbres, du cluster-head vers les autres nœuds du cluster ne sont pas toujours les plus courtes, elles en sont très proches.

Remarque 3 *À l'exception de notre algorithme, basé sur la densité, une diffusion dans un cluster est équivalente à une diffusion dans un environnement fini pour les autres algorithmes de diffusion, contrairement à une diffusion générale qui correspond à une diffusion dans un environnement non borné. On remarquera que tous les protocoles sont plus robustes dans des environnements infinis.*

	500 nœuds		700 nœuds		800 nœuds		900 nœuds		1000 nœuds	
	MOY	MAX	MOY	MAX	MOY	MAX	MOY	MAX	MOY	MAX
MPR	1.76	4.71	1.78	4.85	1.81	4.83	1.81	4.80	1.82	5.00
Densité	1.80	5.08	1.83	5.38	1.87	5.29	1.87	5.50	1.88	5.30

TAB. 4.3 – Temps max et moyen pour recevoir le message dans un cluster. *Les valeurs "MAX" donnent le temps au bout duquel tous les nœuds du réseau ont reçu le message. Les valeurs "MOY" donnent le temps moyen au bout duquel un nœud reçoit le message.*



(a) Clusters (b) Propagation avec MPR (c) Propagation avec les arbres de densité

FIG. 4.7 – Temps de propagation d'un message diffusé dans chacun des clusters représentés sur (a), en utilisant les MPR (b) et notre métrique (c).

4.5.3 Robustesse de la diffusion

Après avoir considéré tous ces résultats, nous nous sommes interrogés sur la robustesse de ces protocoles (toujours en considérant la couche réseau uniquement) envers la cassure de liens. En effet, jusqu'à maintenant, nous n'avons comparé les différents protocoles qu'en terme d'énergie épargnée en limitant les réceptions redondantes et le nombre de ré-émetteurs. Cependant, la redondance apporte de la robustesse au processus de diffusion. Il est donc légitime de se demander si une limitation de la redondance dans un environnement aux liens radios sensibles est bien une bonne approche ou si la redondance en terme de nombre de réceptions a réellement un impact sur la robustesse ? Nous nous sommes aussi interrogés sur l'impact du degré des relais sur cette robustesse : pour une redondance de messages équivalente, est-il préférable d'avoir peu de relais avec un fort degré ou un plus grand nombre de relais avec un plus petit degré ?

Afin d'évaluer cet aspect de la diffusion et de répondre à ces diverses interrogations, nous avons appliqué une probabilité de cassure sur les liens et mesuré la proportion de nœuds qui reçoivent encore le message diffusé. Les simulations que nous avons faites supposent que le message se propage avant qu'aucune information de routage ne soit recalculée par les nœuds comme par exemple l'ensemble des MPR (pour l'heuristique des MPR), l'ensemble des voisins à éliminer (pour les schémas NES) ou le père dans l'arbre de *clustering* (pour l'algorithme basé sur la densité). Comme dans le *blind*

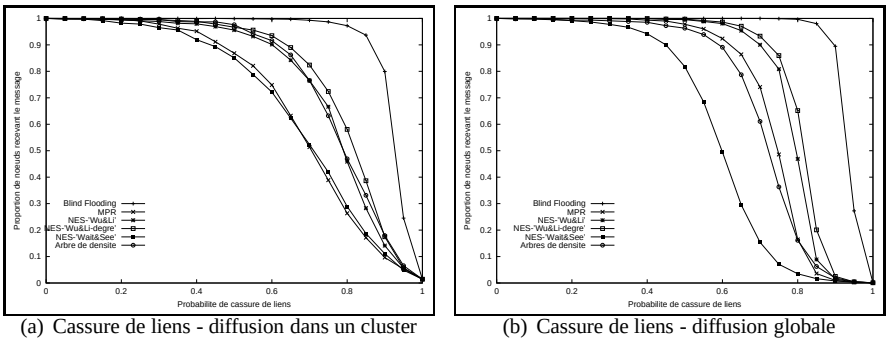
flooding, tous les nœuds retransmettent le message, si des nœuds ne reçoivent pas le message, cela implique que le réseau n'est plus connexe.

La figure 4.8 donne la proportion des nœuds touchés par la diffusion quand on applique une probabilité de cassure sur les liens pour les deux types de diffusion (générale et dans un cluster) et $\lambda = 1000$. Globalement, le comportement des différentes heuristiques est le même pour les deux types de diffusion, excepté pour notre algorithme. Par exemple, le NES-"Wait & See" qui était le meilleur protocole en terme de réceptions et d'émetteurs est le protocole le moins robuste, quel que soit le type de diffusion.

La figure 4.8(a) présente les résultats pour une diffusion dans un cluster. Étonnamment, il ne semble pas que le nombre de réceptions redondantes et la robustesse du protocole soient liés. Par exemple, l'algorithme basé sur les arbres de densité est l'un de ceux induisant le moins de réceptions redondantes et pourtant l'un des plus robustes.

Il semble que les protocoles dont les relais ont un fort degré tendent à être plus robustes. En effet, des protocoles avec des relais à fort degré (NES-degré et Densité) sont très robustes alors que ceux avec des relais à plus faible degré comme les MPR ou le NES-"Wait & See" présentent les pires comportements. Le NES-Wu&Li est moins robuste que sa variante NES-degré (qui augmente le degré des relais). La redondance en terme de nombre de réceptions multiples est donc très coûteuse en terme d'énergie consommée et de bande passante utilisée, sans pour autant apporter plus de robustesse au processus de diffusion.

Notre heuristique présente une bonne robustesse envers les cassures de liens lors d'une diffusion dans un cluster. Comme elle minimise également à la fois le nombre de relais et le nombre de réceptions par nœud, elle constitue le meilleur compromis coût-robustesse-latence.

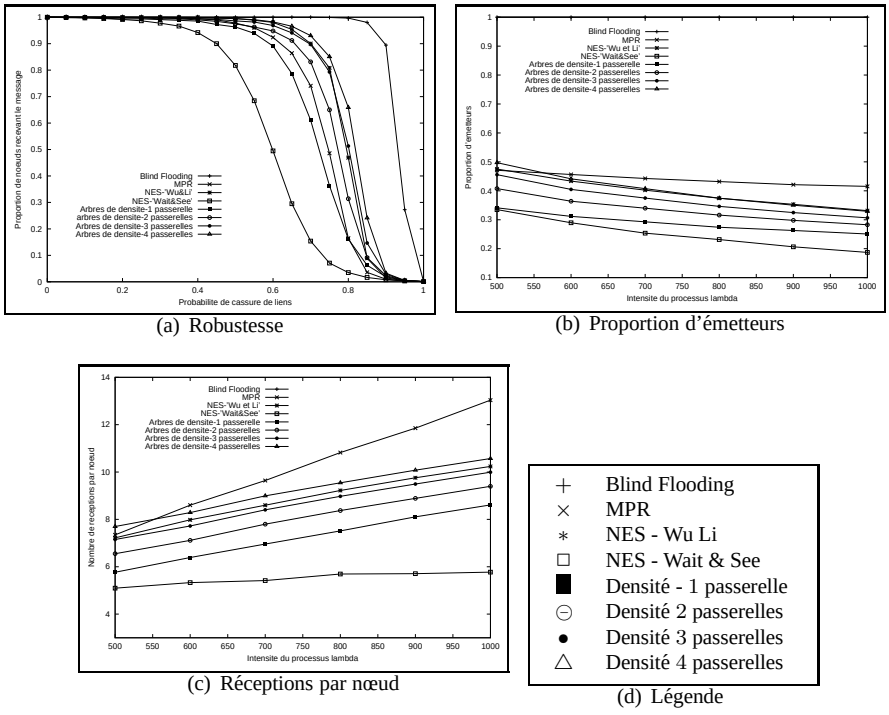


(a) Cassure de liens - diffusion dans un cluster (b) Cassure de liens - diffusion globale

FIG. 4.8 – Proportion de nœuds recevant toujours le message de diffusion après application d'une probabilité de cassure sur les liens lors d'une diffusion dans un cluster (a) ou dans tout le réseau (b).

Cependant, comme le montre la figure 4.8(b), notre algorithme est bien moins robuste lorsqu'il s'agit d'une diffusion globale. Il reste plus robuste que le NES-"Wait & See"

mais bien moins que le protocole NES de Wu et Li. Ce résultat, couplé au fait que notre heuristique est très robuste lorsque le message est propagé dans un cluster seulement, montre que les liens sensibles dans une diffusion générale sont les passerelles. En effet, si un cluster A ne peut être atteint que par le cluster B et que les nœuds de la passerelle de B vers A tombent, l'ensemble du cluster A est alors isolé de la diffusion. Afin d'ajouter de la robustesse à ce niveau, il est donc souhaitable d'élire plusieurs passerelles entre deux clusters voisins. Élire plus d'une passerelle n'ajoute aucun coût lors de la phase d'élection (aucun message supplémentaire n'est utile) mais l'utilisation de passerelles supplémentaires engendre plus d'émetteurs et de réceptions redondantes lors de la propagation d'un message. Il y a donc un compromis à faire entre le nombre de passerelles à élire et le coût de leur utilisation. De façon à estimer ce compromis, nous avons simulé une diffusion d'un message dans le réseau en augmentant le nombre de passerelles utilisées. Les résultats sont donnés par la figure 4.9. L'ajout de passerelles apporte vite de la robustesse au protocole, ce qui confirme le fait qu'il s'agissait bien des liens sensibles. Comme le montre la figure 4.9(a), élire trois passerelles entre chaque paire de clusters voisins permet d'obtenir la même robustesse que pour le protocole NES de Wu et Li, sans pour autant produire plus de réceptions et d'émetteurs que ces heuristiques (figures 4.9(b) et 4.9(c)).



(a) Robustesse (b) Proportion d'émetteurs

(c) Réceptions par nœud (d) Légende

FIG. 4.9 – Robustesse envers les cassures de liens (a), Proportion d'émetteurs (b) et nombre de réceptions par nœud (c).

4.6 Analyse de la sélection des MPR dans OLSR

Comme nous l'avons déjà vu, OLSR est un protocole de routage pro-actif pour les réseaux *ad hoc*, récemment standardisé à l'IETF. Il utilise le concept des Multi-Points Relais (MPR) pour minimiser le trafic de contrôle et calculer les plus courts chemins entre toute paire de nœuds. Chaque nœud du réseau choisit ses MPR dans son voisinage à un saut. Lorsqu'un nœud u reçoit un message M d'un voisin v, il ne le re-transmet que si c'est la première fois qu'il reçoit M et si v a désigné u comme étant un de ses MPR. La sélection des MPR consiste pour un nœud u à choisir un ensemble minimal de nœuds parmi ses voisins de telle façon que l'ensemble des 2-voisins de u soit ainsi couvert (*c.à.d.* que chaque 2-voisin de u reçoit une transmission d'au moins un MPR de u). De cette façon, même si u ne considère que les MPR dans son voisinage, il peut joindre tous ses 2-voisins en 2 sauts, et par extension son k-voisinage en k sauts.

Ce protocole est très célèbre et pourtant, il est loin d'obtenir les meilleurs résultats. Nous nous sommes alors penchés sur la sélection des MPR afin de l'analyser plus en détail et de comprendre pourquoi.

4.6.1 La sélection des MPR

Comme la sélection optimale d'un ensemble minimum de MPR est un problème NP-complet [40], nous donnons ici l'heuristique gloutonne de sélection des MPR qui est celle actuellement utilisée dans l'implémentation d'OLSR.

Pour un nœud $v \in \Gamma(u)$, nous notons $d_u^+(v)$ le nombre de nœuds que u peut atteindre en deux sauts via v :

$$d_u^+(v) = |\Gamma_2(u) \cap \Gamma(v)|$$

Pour un nœud $v \in \Gamma_2(u)$, soit $d_u^-(v)$ le nombre de nœuds dans le voisinage de u qui permettent de connecter u et v en deux sauts :

$$d_u^-(v) = |\Gamma(u) \cap \Gamma(v)|$$

Le nœud u sélectionne dans $\Gamma(u)$ un ensemble de nœuds couvrant intégralement $\Gamma_2(u)$. Cet ensemble est l'ensemble des MPR de u. Nous le notons $MPR(u)$. Chaque nœud a donc son propre ensemble de MPR qui est différent d'un nœud à l'autre. Par définition, $MPR(u)$ est tel que :

$$u \cup \Gamma_2(u) \subset \bigcup_{v \in MPR(u)} \Gamma(v)$$

Considérant un nœud u, nous appelons "nœud isolé" pour u, tout nœud $v \in \Gamma_2(u)$ pour lequel il n'existe qu'un chemin à deux sauts de u à v. En d'autres termes, un nœud v est dit isolé pour u si $d_u^-(v) = 1$.

L'algorithme de sélection des MPR est exécuté sur chaque nœud et suppose que chaque nœud connaît ses voisins à 1 et 2 sauts. Il se décompose en deux étapes. Nous notons

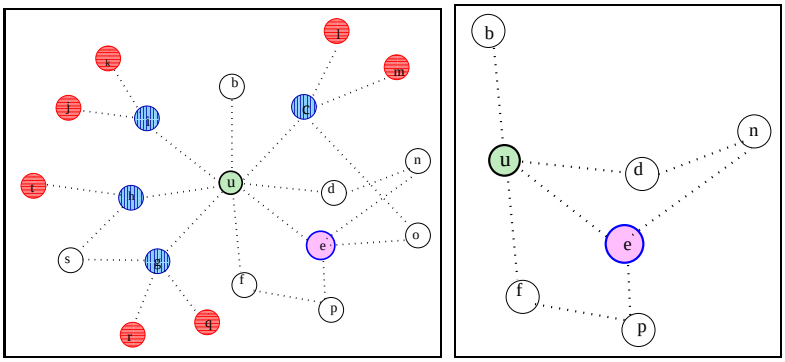
MPR_1 l'ensemble des nœuds MPR sélectionnés lors de la première étape de l'algorithme. Les MPR_1 permettent de couvrir les nœuds isolés. L'algorithme de sélection des MPR est le suivant.

Algorithm 4 Algorithme glouton de sélection des MPR - Exécuté sur chaque nœud.

$\Gamma'(u) = \Gamma(u)$ et $\Gamma'_2(u) = \Gamma_2(u)$.
▷ *Première étape*
Pour tout nœud $v \in \Gamma(u)$
  if ($\exists\, w \in \Gamma(v) \cap \Gamma_2(u) \mid d_u^-(w) = 1$) **then**
    Sélectionne v comme $MPR(u)$.
    ▷ *Sélectionne comme $MPR(u)$, les nœuds de $\Gamma(u)$ couvrant les nœuds "isolés".*
    Retire v de $\Gamma'(u)$ et retire $\Gamma(v) \cap \Gamma_2(u)$ de $\Gamma'_2(u)$.
  end
▷ *Seconde étape*
while ($\Gamma'_2(u) \neq \emptyset$)
  Pour tout nœud $v \in \Gamma'(u)$
    if ($d_u^+(v) = max_{w \in \Gamma'(u)} d_u^+(w)$) **then**
      Sélectionne v comme $MPR(u)$.
      ▷ *Sélectionne comme $MPR(u)$ le nœud v permettant de rattacher le plus de nœuds de $\Gamma_2(u)$ à u en deux sauts.*
      Retire v de $\Gamma'(u)$ et retire $\Gamma(v) \cap \Gamma_2(u)$ de $\Gamma'_2(u)$.
    end

Afin de mieux comprendre cet algorithme, exécutons-le sur le nœud u en vert sur l'exemple de la figure 4.10. Les nœuds isolés apparaissent en rouge, hachurés horizontalement. Par exemple, le nœud t est un nœud isolé pour u car le nœud h est le seul de ses voisins dans $\Gamma(u)$. Le nœud h sera donc élu lors de la première étape de l'algorithme : $h \in MPR_1$. De la même façon, u élira les nœuds bleus, hachurés verticalement h, i, c, g comme MPR_1. Les nœuds $k, j, t, s, r, q, o, m, l$ de $\Gamma_2(u)$ sont ainsi couverts. Le nœud u passe alors à la seconde étape de l'algorithme et ne considère dans $\Gamma_2(u)$ que les nœuds non encore couverts (p et n) et dans Γ_1 les nœuds non MPR_1 (b, f, e et d). Il ne garde donc qu'une vue réduite de la topologie comme illustré sur la figure 4.10(b). Il sélectionne alors son voisin de plus fort degré dans ce graphe. Comme le nœud e couvre n et p alors que f et d ne couvrent chacun qu'un nœud de $\Gamma_2(u)$ (resp. p et n), c'est e qui est élu. A partir de là, tous les nœuds de $\Gamma_2(u)$ sont couverts par les nœuds sélectionnés comme MPR, l'algorithme s'arrête. On a : $MPR(u) = \{c, e, i, h, g\}$.

Plusieurs algorithmes [8, 41, 53] ont été proposés afin d'améliorer cet algorithme et réduire le nombre de MPR sélectionnés. Cependant, aucun d'eux ne réduit considérablement le nombre de MPR. Comme on peut le constater, la première étape de l'algorithme glouton ne peut être supprimée, quel que soit l'algorithme de sélection, puisqu'elle seule permet de couvrir tous les nœuds isolés. De plus, si on veut minimiser le nombre de MPR, cette étape doit être exécutée en premier lieu. C'est pourquoi, toutes les variantes de cet algorithme ne portent en fait que sur la deuxième étape, ce qui laisse in fine une faible marge de manœuvre, comme nous allons le montrer.



(a) Topologie globale - Les nœuds isolés de u apparaissent en rouge et hachurés horizontalement, les MPR_1 de u apparaissent en bleu et hachurés verticalement.

(b) Topologie considérée par u à la fin de la première étape

FIG. 4.10 – Illustration de l'algorithme de sélection des MPR.

4.6.2 Analyse

Nous nous sommes intéressés aux propriétés d'un ensemble MPR sélectionné par un nœud donné. C'est pourquoi, dans notre analyse, nous ne considérons pas l'ensemble du réseau mais un point particulier, ainsi que son voisinage à 1 et 2 sauts. En effet, l'algorithme de sélection de MPR est distribué et exécuté indépendamment par chaque nœud à partir de sa connaissance de son voisinage à 1 et 2 sauts.

Soit $B(x, R)$ la boule de rayon R centrée en x. Nous distribuons un processus ponctuel de Poisson d'intensité $\lambda > 0$ dans $B(0, 2R)$ et ajoutons un point 0 au centre de la boule. Le voisinage de 0 est donc, par définition, l'ensemble des points du processus se trouvant dans $B(0, R)\backslash 0$. C'est pour ce point 0 que nous étudions l'algorithme de sélection des MPR.

Résultats généraux.

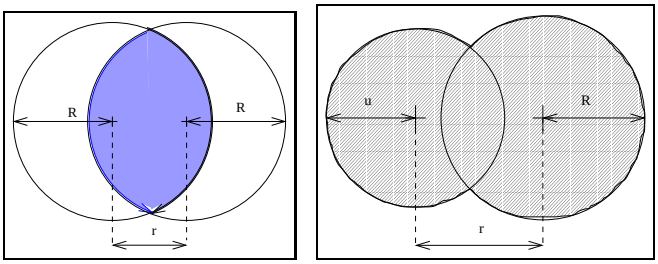
Avant de donner les résultats concernant l'étude des MPR, nous donnons des résultats plus généraux qui nous serviront pour les calculs suivants.

Soit $A(r)$ l'aire de l'intersection de deux boules de rayon R dont les centres sont distants de r (figure 4.11(a)) :

$$A(r) = 2R^2 \arccos\left(\frac{r}{2R}\right) - r\sqrt{R^2 - \frac{r^2}{4}}$$

et $A_1(u, r, R)$ l'aire de l'union de deux disques de rayons respectifs R et u et dont les centres sont distants de r (figure 4.11(b)) :

$$A_1(u, r, R) = rR\sqrt{1 - \left(\frac{R^2 - u^2 + r^2}{2Rr}\right)^2} - R^2 \arccos\frac{u^2 - R^2 - r^2}{2Rr} - u^2 \arccos\frac{R^2 - u^2 - r^2}{2ur}$$



(a) $A(r)$ est l'aire en bleu : aire de l'intersection de deux boules de rayon R.

(b) $A_1(u, r, R)$ est l'aire hachurée : union des aires de deux disques de rayons R et u.

FIG. 4.11 – Illustration des aires $A(r)$ et $A_1(u, r, R)$.

Nous sommes alors en mesure d'évaluer les valeurs moyennes de d_0^+, d_0^-, $|\Gamma(0)|$ et $|\Gamma_2(0)|$ pour une distribution poissonnienne.

Proposition 4 *Soit $u \in \Gamma(0)$ un point uniformément distribué dans $B(0, R)$.*

La valeur moyenne de $d_0^+(u)$ est donnée par :

$$\mathbb{E}\left[d_0^+(u)\right] = \frac{\lambda}{\pi R^2} \int_0^{2\pi} \int_0^R (\pi R^2 - A(r)) r dr d\theta = \lambda R^2 \frac{3\sqrt{3}}{4}$$

Pour obtenir la valeur moyenne de $d_0^+(u)$, l'idée est de compter le nombre moyen de points se trouvant dans l'aire de l'intersection de $B(u, R)$ (1-voisinage de u) et de $B(0, 2R) \backslash B(0, R)$ (2-voisinage de 0) et de sommer pour tout point $u \in \Gamma(0)$.

Soit $v \in \Gamma_2(0)$ un point uniformément distribué dans $B(0, 2R) \backslash B(0, R)$. La valeur moyenne de $d_0^-(v)$ est donnée par :

$$\mathbb{E}\left[d_0^-(v)\right] = \lambda \frac{2}{3R^2} \int_R^{2R} A(r) r dr = \lambda R^2 \frac{\sqrt{3}}{4}$$

Pour obtenir la valeur moyenne de $d_0^-(v)$, on compte le nombre de points se trouvant dans l'intersection de $B(v, R)$ (1-voisins de v) et de $B(0, R)$ (1-voisins de 0) et on somme pour tout point $v \in \Gamma_2(0)$. On remarquera que v peut se trouver dans $B(0, 2R) \backslash B(0, R)$ sans pour autant être un 2-voisin de 0 dans le cas où 0 et v n'ont aucun voisin commun (si $\Gamma(v) \cap \Gamma(0) = \emptyset$). Ainsi, pour obtenir le nombre moyen de 2-voisins de 0, nous devons conditionner le nombre de points v dans $B(R, 2R)$ par la probabilité que v soit un 2-voisin de 0, c.à.d *par la probabilité que $\{\Gamma(v) \cap \Gamma(0) \neq \emptyset\}$. Nous obtenons :*

$$\mathbb{E}\left[d_0^-(v) | v \in \Gamma_2(0)\right] = \frac{\mathbb{E}\left[d_0^-(v)\right]}{\mathbb{P}(d_0^-(v) > 0)}$$

où

$$\mathbb{P}\left(d_0^-(v) > 0\right) = 1 - \frac{2}{3R^2}\int_R^{2R} exp\{-\lambda A(r)\} r dr$$

La taille moyenne du voisinage de 0 *est donnée par :*

$$\tilde{\delta}(0) = \mathbb{E}\left[|\Gamma(0)|\right] = \lambda\pi R^2$$

Le nombre moyen de voisins à deux sauts de 0 *(*$|\Gamma_2(0)|$*) correspond au nombre de points* v *du processus se trouvant dans* $B(0,2R)\backslash B(0,R)$*, conditionné par la probabilité qu'il existe un voisin commun à chaque* v *et* 0*. On obtient :*

$$\begin{aligned}\mathbb{E}\left[|\Gamma_2(0)|\right] &= 3\lambda\pi R^2 \mathbb{P}\left(d_0^-(v) > 0\right) \\ &= 3\lambda\pi R^2 \left(1 - \frac{2}{3R^2}\int_R^{2R} exp\{-\lambda A(r)\} r dr\right)\end{aligned}$$

Analyse de la première étape de l'algorithme de sélection des MPR.

Nous nous intéressons maintenant plus particulièrement à la première étape de l'algorithme de sélection. Dans un premier temps, nous déterminons le nombre moyen de points isolés pour le point 0. Comme nous l'avons vu, l'unique voisin d'un point isolé appartenant aussi à $\Gamma(0)$ est obligatoirement un MPR_1. Cependant, le nombre de points isolés ne nous donne pas directement le nombre de MPR_1 mais une borne supérieure car un même MPR_1 peut couvrir plusieurs points isolés. Par exemple, sur la figure 4.10(a), nous avons quatre MPR_1 mais sept nœuds isolés. Le MPR_1 i permet de couvrir deux nœuds isolés : les nœuds j et k.

Par définition, les nœuds isolés sont les nœuds $v \in \Gamma_2(0)(u)$ tels que $d_0^-(v) = 1$.

Proposition 5 *Soit* v *un point uniformément distribué dans* $B(0,2R)\backslash B(0,R)$ *et* D *l'ensemble des points isolés* v *(tels que* $d_0^-(v) = 1$*). On obtient :*

$$\mathbb{P}\left(d_0^-(v) = 1\right) = \frac{2}{3R^2}\int_R^{2R} \lambda A(r) exp\{-\lambda A(r)\} r dr$$

Tout comme dans la proposition 4, nous ne considérons que les nœuds v *tels que* $v \in \Gamma_2(0)$ *:*

$$\mathbb{P}\left(d_0^-(v) = 1 | v \in \Gamma_2(0)\right) = \frac{\mathbb{P}\left(d_0^-(v) = 1\right)}{\mathbb{P}\left(d_0^-(v) > 0\right)}$$

Nous pouvons alors déduire de cette probabilité le nombre moyen de points isolés :

$$\mathbb{E}\left[|D|\right] = 2\pi\lambda^2 \int_R^{2R} A(r) exp\{-\lambda A(r)\} r dr$$

qui constitue une borne supérieure pour le nombre de MPR_1 *:*

$$\mathbb{E}\left[|MPR_1|\right] \leq \mathbb{E}\left[|D|\right]$$

Dans la proposition suivante, nous donnons une borne inférieure du nombre moyen de MPR_1 :

Proposition 6 *Soit u un point uniformément distribué dans $B(0,R)$.*

$$\mathbb{P}\left(u \in MPR_1\right) \geq \frac{2}{R^2}\mathbb{P}\left(d_0^+(u)>0\right)\int_0^R\int_R^{R+r} f(x,r,R)\exp\left\{-\lambda\left(2\pi R^2 - A_1(R,x,R)\right)\right\}rdxdr$$

où $f(x,r,R)$ est la fonction densité de probabilité :

$$f(x,r,R) = -\frac{\lambda\left[\frac{\partial}{\partial x}A_1(x,r,R) - 2\pi x\right]}{1-\exp\left\{-\lambda\left(A_1(R,r,R)-\pi R^2\right)\right\}}\exp\left\{-\lambda\left(A_1(x,r,R)-\pi x^2\right)\right\}$$

De cette probabilité, nous pouvons déduire une borne inférieure pour le nombre moyen de MPR_1 :

$$\mathbb{E}\left[|MPR_1|\right] \geq 2\lambda\pi\mathbb{P}\left(d_0^+(u)>0\right)\int_0^R\int_R^{R+r} f(x,r,R)\exp\left\{-\lambda\left(2\pi R^2 - A_1(R,x,R)\right)\right\}rdxdr$$

Preuve 6 *Pour établir la borne inférieure de la probabilité qu'un point de $\Gamma(0)$ soit un MPR_1, nous nous basons sur une condition suffisante. Une condition suffisante pour que $u \in MPR_1$ est que le point w de $\Gamma(u)$ le plus éloigné de 0 est un point isolé ($d_0^-(w)=1$).*

Connaissant r, la distance entre 0 et u, on est capable de calculer la distribution de la distance entre 0 et w (le point du voisinage de u qui est le plus éloigné de 0). Sachant alors la distance x entre 0 et w, on calcule la probabilité qu'il n'existe qu'un seul point (le point u) dans l'intersection des voisinages de 0 et w. Cette dernière condition garantit que w est un point isolé et que $u \in MPR_1$. On intègre alors cette dernière probabilité par les distributions de r et de x pour obtenir le résultat final. Cette borne est très fine puisque, dans la plupart des cas, les points isolés sont les nœuds les plus éloignés de 0. ■

Nous nous sommes aussi intéressés à la distribution spatiale des MPR_1. Nous avons pour cela calculé un encadrement de la probabilité qu'un point $u \in \Gamma(0)$ à distance r de 0 soit un MPR_1 en fonction de sa distance à 0. Pour cela, nous considérons un point $u \in \Gamma(0)$ à distance r ($0 < r \leq R$) de 0. Nous fixons ces deux points (u et 0) et distribuons les points du processus de Poisson dans $B(0,2R)$ indépendamment de ces deux points. Nous cherchons alors quelle est la probabilité que ce nœud soit un $MPR_1(0)$. Dans la proposition suivante, nous proposons un encadrement de cette probabilité.

Proposition 7 *Soit u un point à distance r ($0 < r \leq R$) de 0.*

$$\mathbb{P}\left(u \in MPR_1\right) \geq \left(1 - \exp\left\{-\lambda(\pi R^2 - A(r))\right\}\right) \int_R^{R+r} f(v, r, R) \exp\left\{-\lambda(2\pi R^2 - A_1(R, v, R))\right\} dv$$

$$\mathbb{P}\left(u \in MPR_1\right) \leq 1 - \left(1 - \exp\left\{-\lambda \frac{A(R+r)}{2}\right\}\right)^2$$

Preuve 7 *La borne inférieure est obtenue de la même façon que la borne inférieure du nombre de MPR_1 donnée dans la proposition 6, mais la distance entre 0 et u est cette fois fixée. La borne supérieure est obtenue à partir de l'idée suivante. S'il existe des points v dans les deux demi-intersections de cercle illustrés sur la figure 4.12, la plupart des voisins de u appartenant à $\Gamma(u) \cap \Gamma_2(0)$ sont couverts par ces points v (en plus de u) et ne sont donc pas isolés. Concernant les points non couverts par les points v, nous pouvons montrer facilement que la même borne reste valable. Cela nous donne une probabilité que u ne soit pas un MPR_1, de laquelle nous pouvons déduire la probabilité que u soit un MPR_1.* ■

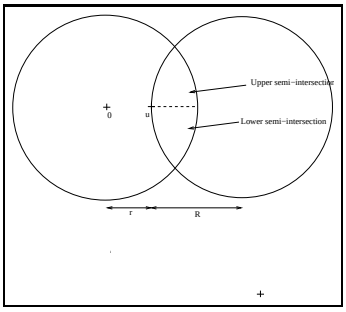
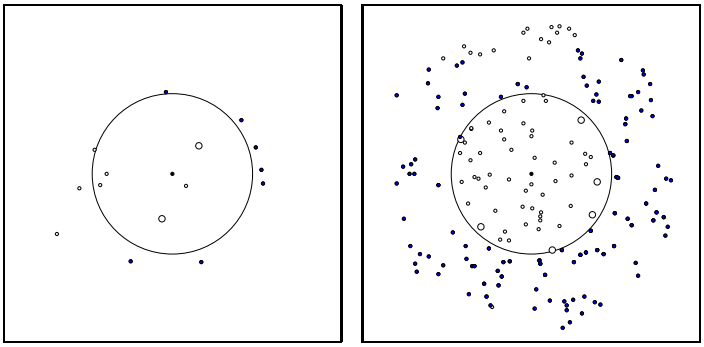


FIG. 4.12 – Les deux demi-intersections utilisées dans la preuve de la proposition 7.

4.6.3 Résultats numériques et simulations

Afin d'estimer de façon numérique les résultats obtenus précédemment, nous avons simulé l'algorithme de sélection des MPR. Nous utilisons le même modèle que celui étudié lors de l'analyse théorique, à savoir que les nœuds sont répartis sur une boule $B(0, 2)$ ($R = 1$) avec un processus ponctuel de Poisson d'intensité $\lambda > 0$. Nous ajoutons le point 0 au centre de la boule et étudions le nombre de MPR sélectionnés par ce point à chaque étape de l'algorithme. La figure 4.13 représente des échantillons du modèle pour différentes valeurs de λ. Le point 0 pour lequel nous étudions l'algorithme est le point noir central. Les points à l'intérieur de cercle sont les points de $\Gamma(0)$, les

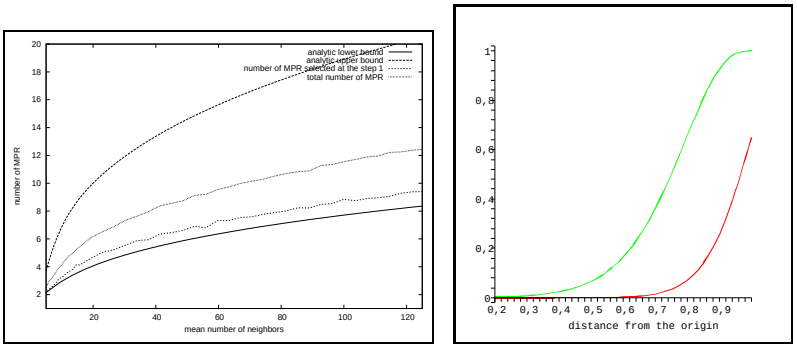


(a) $\lambda\pi = 6$ (b) $\lambda\pi = 45$

FIG. 4.13 – Sélection des MPR pour $\lambda\pi = 6$ et $\lambda\pi = 45$.

plus gros étant les MPR_1. Les points à l'extérieur du cercle sont les voisins à deux sauts de 0, les points bleus étant ceux couverts par les MPR_1.

On remarquera que dans les deux cas, la quasi totalité du 2-voisinage de 0 est couvert par les MPR_1. L'ajout d'un seul point MPR suffirait à couvrir l'intégralité de $\Gamma_2(0)$. Nous avons vu qu'il existe en moyenne un grand nombre de points isolés, donnant naissance à un grand nombre de MPR_1. Ces MPR_1 semblent être régulièrement distribués près de la frontière de $B(0, R)$, ce qui confirme les résultats obtenus dans la proposition 7 et explique pourquoi ils couvrent une grande partie de $\Gamma_2(0)$.



(a) Nombre moyen de MPR et MPR_1 obtenus par simulation et bornes analytiques du nombre de MPR_1.

(b) Bornes inférieure et supérieure de la probabilité pour un voisin de 0 d'être un $MPR_1(0)$ en fonction de la distance au point 0.

FIG. 4.14 – Comparaison des résultats analytiques et de simulation.

La figure 4.14(a) montre le nombre moyen de MPR et MPR_1 obtenus par simulation ainsi que les bornes analytiques du nombre de MPR_1. On observera qu'approximativement 75% des MPR sont élus lors de la première étape de l'algorithme (et sont

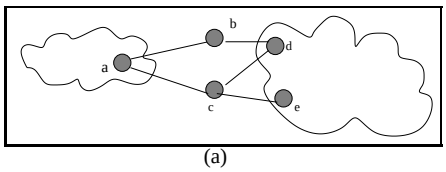
des MPR_1), ce qui confirme le fait que les MPR_1 couvrent la quasi-totalité du 2-voisinage. Comme mentionné auparavant, la borne inférieure est une borne très fine du nombre de MPR_1 moyen.

L'encadrement de la probabilité qu'un voisin de 0 soit élu comme $MPR_1(0)$ (donné dans la proposition 7), nous permet de montrer que les MPR_1 sont répartis à proximité de la frontière de la portée radio R de 0. La figure 4.14(b) montre ces bornes pour une distance r entre les nœuds 0 et ses voisins variant de 0.2 à 0.999 pour $\lambda = 15$. On remarque que ces résultats dépendent de λ : plus λ augmente, plus la distance entre 0 et ses MPR_1 augmente aussi (puisque 0 a plus de chances d'avoir des voisins proches de la frontière).

4.6.4 Conséquences

Comme nous l'avons vu dans les sections précédentes, le but recherché en introduisant les MPR est de minimiser le nombre de relais lors de la diffusion du trafic de contrôle. Le nombre de MPR doit donc être aussi petit que possible. Bien que plusieurs travaux aient cherché à optimiser l'algorithme glouton de sélection des MPR, seule la seconde étape de l'algorithme peut être améliorée puisque la première est indispensable pour couvrir l'ensemble du 2-voisinage d'un nœud. Or, comme nous avons pu le constater au cours de nos analyses et simulations, cette première étape mène à la sélection de plus de 75% des MPR. Cela signifie que les améliorations pouvant être apportées ne portent que sur 25% des MPR, ce qui explique que les variantes de l'algorithme de sélection ne produisent aucune amélioration significative.

Malheureusement, cette caractéristique peut également être une source de problème de robustesse. En effet, si 75% des MPR de u couvrent au moins un nœud isolé, la perte d'un de ces nœuds engendre une forte probabilité qu'au moins un voisin v à 2 sauts de u ne reçoive plus un message envoyé u. Il se peut bein sûr que v reçoive le message de u via un autre chemin mais ce dernier serait plus optimal comme le clame OLSR. De plus, si v est tel que $v \in MPR(u)$ et $v \notin MPR(w)$, il peut recevoir un message pour la première fois par w plutôt que u mais ne le re-transmettra pas puisqu'il n'est pas un MPR de w. Cela peut conduire à l'isolation de certaines parties du réseau lors d'une diffusion, comme l'illustre la figure 4.15. Les nuages représentent deux parties connexes du réseau, connectées par les nœuds b et c. Comme le nœud e est un nœud isolé pour a, a élit c en tant que MPR. Il ne choisit pas b puisque le nœud d couvert par b l'est également par c. Supposons que le lien entre c et a tombe et que la diffusion se propage avant que a n'ait pu recalculer ses MPR. Bien que le réseau soit encore connecté, la partie droite du réseau ne sera pas touchée par la diffusion car b n'étant pas MPR de a, il ne re-transmettra pas le message. Ce phénomène peut expliquer les mauvais résultats obtenus par les MPR lors de l'étude de la robustesse des protocoles de diffusion en section 4.5. De plus, les liens entre un nœud et ses MPR ont de fortes probabilités de casser dans un environnement mobile puisque, comme nous avons pu le constater, les MPR sont situés à proximité de la frontière de la portée de transmission des nœuds. Ils sont donc plus enclins à basculer en dehors de la zone de transmission et ainsi à casser le lien radio.



(a)

FIG. 4.15 – Exemple.

4.7 Conclusion et perspectives

Dans ce chapitre, nous avons tiré avantage de certaines caractéristiques des clusters formés par notre heuristique pour proposer une utilisation supplémentaire de la structure. En effet, la structure d'arbres sous-jacente des clusters permet l'application d'un protocole de diffusion aussi bien dans un cluster que sur l'ensemble du réseau, ceci avec un coût faible et borné et une maintenance quasi-locale. Nous avons analysé de façon théorique le nombre de réceptions par nœud lors d'une diffusion. Puis, nous avons pu constater que notre algorithme de diffusion proposait le meilleur compromis coût en énergie - latence - robustesse parmi les protocoles de diffusion existant dans la littérature.

Dans le futur, il serait intéressant d'étudier ce protocole dans un environnement plus mobile. Dans notre approche, nous considérons une couche MAC idéale afin de pouvoir comparer les protocoles de niveau 3 sans s'occuper des problèmes qui peuvent survenir aux niveaux inférieurs et influer sur les performances de chacun de ces protocoles. Cependant, comme nous l'avons mentionné, plus le nombre de messages échangés est important, plus fortes sont les collisions survenant aux niveaux inférieurs. Il semblerait donc intéressant d'étudier un protocole de diffusion qui ne soit pas cloisonné à la couche réseau mais qui prenne en considération les caractéristiques de plusieurs couches en même temps, plutôt que de chercher à optimiser un protocole à un niveau particulier.

4.8 Publications

1. **Journaux et revues avec comité de lecture :**
   (a) *Efficient Broadcasting in Self-Organizing Sensor Networks.* Nathalie Mitton, Anthony Busson et Éric Fleury. International Journal of Distributed Sensor Networks (IJDSN), Volume 1, Janvier 2006.
2. **Colloques et conférences internationaux avec comité de lecture :**
   (a) *Efficient Broadcasting in Self-Organizing Multi-Hop Wireless Network.* Nathalie Mitton, Éric Fleury. Conference on AD-HOC Networks & Wireless (Ad Hoc Now'05), 6-8 Octobre 2005, Cancùn, Mexique. *Sélectionné parmi les meilleurs papiers pour une soumission à une issue spéciale du journal JDA.*
   (b) *An analysis of the MPR selection in OLSR and consequences*. Anthony Busson, Nathalie Mitton and Éric Fleury. Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET'05), Juin 2005, Ile de Porquerolles, France.
   (c) *Broadcast Analysis in Multi-hop Wireless Networks*. Nathalie Mitton, Anthony Busson and Éric Fleury. Invited Paper at Spatial Stochastic Modeling of Wireless Networks (SpasWIN'05), Avril 2005, Riva de Garda, Italie.
   (d) *An analysis of the MPR selection in OLSR*. Anthony Busson, Nathalie Mitton and Éric Fleury. Spatial Stochastic Modeling of Wireless Networks (SpasWIN'05), Avril 2005, Riva de Garda, Italie.
3. **Colloques et conférences nationaux :**
   (a) *Une analyse de la sélection des MPR dans OLSR*. Anthony Busson, Nathalie Mitton et Éric Fleury. ALGOTEL'05, Mai 2005, Presqu'île de Giens, France.
4. **Rapports de recherche :**
   (a) *Broadcast in Self-organizing Wireless Multi-hop Network*. Nathalie Mitton, Anthony Busson and Éric Fleury. RR-5487. Février 2005.
   (b) *An analysis of the MPR selection in OLSR*. Anthony Busson, Nathalie Mitton and Éric Fleury. RR-5468. Janvier 2005.
5. **Journaux en cours de soumission :**
   (a) *Efficient Broadcasting and Self-Stabilization in Self-Organizing Multi-hop Wireless Networks.* Nathalie Mitton, Éric Fleury, Isabelle Guérin-Lassous and Bruno Séricola and Sébastien Tixeuil. "Best Papers of Adhoc Now 2005" special issue of the Journal of Discrete Algorithms (JDA).
6. **Séminaires, présentations, exposés :**
   (a) *Diffusion efficace dans les réseaux sans fil multi-sauts.* Nathalie Mitton, Éric Fleury. Journées RESCOM - Villeneuve d'Ascq - France - 6-7 Mars 2005.
   (b) *An analysis of the MPR selection in OLSR.* Anthony Busson, Nathalie Mitton, Eric Fleury. Séminaire ACI FRAGILE - Aussois - France - 23-24 Mars 2005.

4.9 Annexes

Nous présentons ici les preuves des propositions 1 et 2 donnant le nombre de réceptions par nœud lors de la diffusion d'un message.

Preuve de la proposition 1 Étant donné un processus ponctuel stationnaire Φ d'intensité λ ($\lambda > 0$), soit Φ_{Relay} d'intensité λ_{Relay} un amincissement de Φ. Les points de Φ_{Relay} représentent les relais. Nous supposons que Φ_{Relay} est toujours un processus ponctuel stationnaire. On cherche à montrer que le nombre moyen de réceptions d'un même message par nœud $\overline{r}$ est :

$$\overline{r} \quad = \quad \mathbb{E}^{o}_{\Phi_{Relay}} \left[\Phi(B_0^{'}) \right]$$

où $\mathbb{E}^{o}_{\Phi} \left[\Phi_{Relay}(B_0^{'}) \right]$ est l'espérance sous Palm par rapport au processus Φ (et donc la valeur moyenne) du nombre de relais dans $B_0^{'}$.

Pour un point donné, *c.à.d.* le point 0 sous les probabilités de Palm, le nombre moyen de réceptions correspond au nombre moyen de points de Φ_{Relay} dans le voisinage de 0 (à distance inférieure ou égale à R).

À partir de la formule de Mecke [76], on peut déduire que le nombre total de réceptions Z reçues par l'ensemble des nœuds d'une surface S est :

$$Z = \mathbb{E}\left[\int_S \Phi_{Relay}(B_x^{'}) \Phi(dx) \right] = \lambda \mathbb{E}^{o}_{\Phi} \left[\Phi_{Relay}(B_0^{'}) \right]$$

Par stationnarité des deux processus ponctuels Φ et Φ_{Relay}, nous avons :

$$\mathbb{E}\left[\int_S \Phi_{Relay}(B_x^{'}) \Phi(dx) \right] = \mathbb{E}\left[\int_S \Phi(B_x^{'}) \Phi_{Relay}(dx) \right]$$

La partie gauche de l'équation est le nombre total de réceptions reçues par les nœuds de la surface S, sachant que les nœuds en bordure peuvent recevoir le message depuis des nœuds en dehors de S. La partie droite de l'équation est le nombre de réceptions reçues par l'ensemble des nœuds du processus Φ mais générées uniquement par les relais se trouvant dans S. En appliquant la formule de Mecke de part et d'autre de l'équation, on obtient :

$$\lambda \mathbb{E}^{o}_{\Phi} \left[\Phi_{Relay}(B_0^{'}) \right] = \lambda_R \mathbb{E}^{o}_{\Phi_{Relay}} \left[\Phi(B_0^{'}) \right]$$

d'où :

$$\overline{r} = \mathbb{E}^{o}_{\Phi} \left[\Phi_{Relay}(B_0^{'}) \right] = \mathbb{E}^{o}_{\Phi_{Relay}} \left[\Phi(B_0^{'}) \right] \mathbb{P}^{o}_{\Phi} \left(0 \in \Phi_{Relay} \right) = \frac{\lambda_{Relay}}{\lambda} \mathbb{E}^{o}_{\Phi_{Relay}} \left[\Phi(B_0^{'}) \right]$$

Preuve de la proposition 2 Étant donné un graphe aléatoire $G(V, E)$ et un ensemble de relais $Relay \subset V$ où les degrés des nœuds et des relais ainsi que le nombre de réceptions par nœud sont équi-distribués. On cherche à montrer que le nombre moyen de réceptions par nœud $\overline{r}$ s'écrit :

$$\overline{r} \quad = \quad \mathbb{E}\left[\delta(v_1)\middle| v_1 \in Relay\right]\mathbb{P}(v_1 \in Relay) \tag{4.2}$$

Soient N une variable aléatoire représentant le nombre de sommets dans G ($N = |V|$) et Z le nombre total de réceptions induites sur les nœuds du réseau par la diffusion. Pour tout $u \in V$, nous définissons $\delta_R(u)$ comme le nombre de relais dans le voisinage de u.

Comme seuls les relais émettent le message, les liens étant bidirectionnels, le nombre de réceptions du message perçues par un nœud (qu'il soit lui-même relais ou non) correspond au nombre de relais dans son voisinage. Nous avons donc :

$$\overline{r} = \mathbb{E}[\delta_R(u)],\ \forall u \in V$$

où $\mathbb{E}[\delta_R(u)]$ est l'espérance de la variable $\delta_R(u)$ et correspond à sa valeur moyenne.

Les liens étant bi-directionnels, Z peut s'écrire de deux façons :

$$Z = \sum_{u \in V} \delta_R(u) \tag{4.3}$$

ou

$$Z = \sum_{v \in Relay} \delta(v) = \sum_{v \in G} \delta(v)\mathbb{1}_{v \in Relay} \tag{4.4}$$

où $\mathbb{1}_{v \in Relay} = 1$ if $v \in Relay$ et $\mathbb{1}_{v \in Relay} = 0$ sinon.

À partir de la première formulation de Z (équation 4.3), on a

$$\begin{aligned}
\mathbb{E}\left[\frac{Z}{N}\right] &= \sum_{k=1}^{+\infty} \mathbb{E}\left[\frac{\sum_{i=1}^{k}\delta_R(u_i)}{k}\middle| N = k\right]\mathbb{P}(N = k) \\
&= \sum_{k=1}^{+\infty}\sum_{i=1}^{k}\frac{1}{k}\mathbb{E}\left[\delta_R(u_i)\middle| N = k\right]\mathbb{P}(N = k) \\
&= \sum_{k=1}^{+\infty}\frac{k}{k}\mathbb{E}\left[\delta_R(u_1)\middle| N = k\right]\mathbb{P}(N = k) \\
&= \sum_{k=1}^{+\infty}\mathbb{E}\left[\delta_R(u_1)\middle| N = k\right]\mathbb{P}(N = k) \\
&= \mathbb{E}\left[\delta_R(u)\right]
\end{aligned}$$

Couplé à la définition de $\overline{r}$ donné en équation 4.2, on a :

$$\overline{r} \quad = \quad \mathbb{E}\left[\frac{Z}{N}\right]$$

Cette dernière égalité nous permet de calculer la valeur moyenne de $\frac{Z}{N}$: $\mathbb{E}\left[\frac{Z}{N}\right]$ en utilisant cette fois la deuxième formulation de Z (équation 4.4). Nous conditionnons cette quantité par les différentes valeurs de N :

$$\begin{aligned}
\overline{r} &= \mathbb{E}\left[\frac{Z}{N}\right] \\
&= \mathbb{E}\left[\frac{\sum_{v\in V}\delta(v)\mathbb{1}_{v\in Relay}}{N}\right] \\
&= \sum_{k=1}^{+\infty}\mathbb{E}\left[\frac{\sum_{i=1}^{k}\delta(v_i)\mathbb{1}_{v_i\in Relay}}{k}\Big|N=k\right]\mathbb{P}(N=k) \\
&= \sum_{k=1}^{+\infty}\sum_{i=1}^{k}\frac{1}{k}\mathbb{E}\left[\delta(v_i)\mathbb{1}_{v_i\in Relay}\right]\mathbb{P}(N=k) \\
&= \sum_{k=1}^{+\infty}\mathbb{E}\left[\delta(v_1)\mathbb{1}_{v_1\in Relay}\Big|N=k\right]\mathbb{P}(N=k) \\
&= \mathbb{E}\left[\delta(v_1)\mathbb{1}_{v_1\in Relay}\right] \\
&= \mathbb{E}\left[\delta(v_1)\Big|v_1\in Relay\right]\mathbb{P}(v_1\in Relay)
\end{aligned}$$

où v_1 est un nœud choisi arbitrairement parmi les sommets de G.

Chapitre 5

Localisation et routage

5.1 Introduction

Nous avons proposé un algorithme de *clustering* pour organiser le réseau (chapitre 3) afin de pouvoir utiliser le réseau sans fil multi-sauts sur de larges échelles. Nous avons vu comment une telle structure de clusters peut être utilisée pour effectuer une diffusion efficace, aussi bien dans tout le réseau que dans un cluster. Dans ce chapitre, nous expliquons comment nous comptons utiliser notre organisation de clusters pour le routage et permettre à toute paire de nœuds de communiquer. Dans tout type de réseau, pour router un message vers un nœud destination v, un nœud u doit avoir une information sur la position de v. Dans les réseaux filaires, l'information de routage est encapsulée dans l'adresse du nœud, celle-ci étant dépendante de la topologie du réseau. Par exemple, une adresse IP identifie un nœud et en même temps permet de le situer puisque le préfixe du réseau est inclus dans l'adresse IP. Dans les réseaux sans fil, l'identifiant permanent du nœud ne peut inclure sa position du fait de sa mobilité et est donc indépendant de la topologie sous-jacente. Les protocoles de routage utilisés dans les réseaux filaires ne peuvent être appliqués. Une approche possible est d'utiliser un routage indirect. Une opération de routage est qualifiée de *indirecte* si elle s'effectue en deux étapes : *(i)* le look-up qui permet de **situer** le nœud cible, puis, *(ii)* le routage qui permet à la source de **communiquer** directement avec le nœud ciblé. La figure 5.1 illustre un routage indirect. Le nœud u veut communiquer avec le nœud v mais il doit d'abord le localiser. Pour cela, il effectue l'opération de look-up : il demande à une troisième entité (ici, le nœud w) où se trouve v. Cette entité est un point de rendez-vous : v enregistre régulièrement sa position auprès de w qui garde une trace de la position de v. Une fois que w a répondu à u, u est en mesure de contacter directement v. Ce principe de routage est par exemple utilisé dans les réseaux de téléphonie GSM[1] ou dans le protocole Mobile IP[2], où la position de la destination est préalablement demandée respectivement aux HLR (Home Location Register) ou aux Home Agent avant

[1] http ://www.gsm.org
[2] http ://www.ietf.org/rfc/rfc2002.txt

d'établir directement la communication entre le demandeur et la destination. Cependant, un tel principe ne peut être mis en œuvre dans un réseau sans fil de type *ad hoc* car tous les nœuds sont susceptibles de bouger, y compris les Home Agents potentiels.

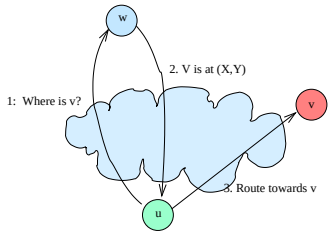


FIG. 5.1 – Routage indirect : le nœud u veut communiquer avec le nœud v mais ne connaît pas sa position. Il demande donc dans un premier temps à une troisième entité (ici, le nœud w) où se trouve v. w sait où se trouve v car v enregistre régulièrement sa position auprès de w qui peut répondre à u. u est alors en mesure de contacter v.

Nous nous proposons d'appliquer un schéma de routage indirect pour router dans des réseaux sans fil multi-sauts à large échelle. Nous désirons une solution qui permette le passage à l'échelle du réseau et qui doit donc maintenir le moins d'information possible sur les nœuds. Nous voulons également éviter les situations où la distance entre la source (le nœud u sur la figure 5.1) et le point de rendez-vous (le nœud w) est plus importante que la distance entre la source (le nœud u) et la destination (le nœud v). En effet, si la requête de localisation doit traverser deux fois le réseau avant qu'une communication entre deux nœuds qui peuvent être proches s'établisse, nous occupons inutilement la bande passante et introduisons une forte latence, ce qui empêche le protocole d'être extensible.

Un moyen d'appliquer un routage indirect est d'utiliser une Table de Hachage Distribuée (DHT - *Distributed hash table*). L'implémentation des DHT dans les réseaux sans fil a donné naissance à de nouvelles problématiques [64]. De plus, un tel adressage offre une approche prometteuse pour permettre le passage à l'échelle [30]. Les DHT fournissent une association générale entre une clef et toute sorte d'information (comme l'identité d'un nœud ou une position). Elles utilisent un espace d'adressage virtuel $\mathcal{V}$. Des partitions de cette espace virtuel sont allouées aux nœuds du réseau. L'idée est d'utiliser une fonction depuis un espace réel vers un espace virtuel $\mathcal{V}$. Cette fonction, dite de $hash$, permet aux nœuds d'identifier certains points de rendez-vous auprès desquels ils enregistrent leur position. Cette fonction de $hash$ est connue de tous les nœuds du réseau et peut ensuite être utilisée par un nœud source pour retrouver ces mêmes points de rendez-vous et leur demander la position du nœud qu'ils recherchent. Une information connue de tous (comme le nom de notre destinataire) est *hashée* en une clef ($hash(v) = cle_v$) de l'espace d'adressage virtuel $\mathcal{V}$. Les informations associées à cette clef (comme la position des nœuds) sont ensuite stockées sur le (ou les) nœud(s) responsable(s) de la partition de l'espace virtuel auquel la clef appartient. Par exemple, sur la figure 5.1, nous avons $hash(v) = Position\_nœud\_v \in I \subset \mathcal{V}$ et le nœud w est responsable de l'intervalle I. En connaissant I, les nœuds v et u sont

capables de trouver w soit pour s'enregistrer (pour le nœud v) soit pour lui demander où se trouve v (pour le nœud u). On remarque que les nœuds u et v n'ont pas besoin de connaître la vraie identité de w, mais juste son adresse virtuelle dans $\mathcal{V}$. Cette opération retournant le(s) nœud(s) responsable(s) d'une certaine clef dans les systèmes utilisant des DHT, est appelée *look-up*. Plus de détails au sujet des opérations de look-up sont donnés dans [10].

Dans la littérature, les DHT sont utilisées à deux niveaux : au niveau de la couche application et au niveau de la couche réseau. Les DHT sont utilisées au niveau applicatif en particulier dans les systèmes pair-à-pair. La clef "hachée" dans ces systèmes de partage de fichiers est l'identifiant d'un fichier ($cle = hash(fichier)$). L'information associée à la clef et maintenue par le nœud responsable de cette clef est l'identité des nœuds du réseau qui détiennent ce fichier. Les adresses virtuelles des nœuds (ou partitions de $\mathcal{V}$ dont ils sont responsables) forment un réseau *overlay* (*c.à.d* un réseau virtuel basé sur le réseau physique qui maintient des liens logiques entre les nœuds) sur lequel les requêtes sont routées. Ce qui différencie majoritairement les différentes propositions de réseaux pair-à-pair dans la littérature est la géométrie de ce réseau *overlay*. En effet, la forme de ce réseau va de l'anneau (Chord [73]) au graphe de De Bruijn (D2B [33]) en passant par des arbres (Tapestry [84], Kademlia [54]), des espaces d-dimensionnels (CAN [68]), des structures en forme de papillon [52] ou encore des structures hybrides arbres-anneaux (Pastry [70]).

Lorsqu'elles sont utilisées au niveau de la couche réseau, les DHT distribuent les informations de position des nœuds à travers le réseau et sont utilisées pour identifier un nœud pouvant fournir des informations permettant de joindre le nœud destination. C'est de cette façon que nous nous proposons d'utiliser les DHT. Quand un nœud u doit envoyer une information à v, il doit d'abord le localiser et pour cela, demander à un nœud w, responsable de la clef $k = hash(v)$. Parmi les DHT appliquées au niveau réseau, le routage utilisé dans les différentes propositions est de deux sortes : un routage indépendant de la DHT (le routage n'utilise pas l'espace virtuel $\mathcal{V}$) et un routage dépendant de la DHT (le routage utilise l'espace virtuel $\mathcal{V}$).

Lorsque le routage est indépendant de la DHT, les nœuds disposent généralement de leur coordonnées géographiques absolues (obtenues par exemple avec un GPS) ou relatives (comme dans [17]). C'est cette information qu'ils associent à la clef. En effectuant $hash$(destination), un nœud u obtient des coordonnées géographiques d'une "zone rendez-vous" $\mathcal{A}$. u peut alors appliquer un routage géographique classique afin d'envoyer sa requête vers un nœud v se trouvant dans la zone $\mathcal{A}$ et qui détient les coordonnées géographiques de la destination cherchée par u. De là, u effectue de nouveau un routage géographique mais cette fois, directement vers la destination. Ce type de routage indépendant de la DHT est utilisé par exemple dans [6, 57, 58], dans les projets "Terminodes" [15, 16] et "Grid" [48]. Dans notre cas, les nœuds ne disposent d'aucune information concernant leur position géographique et nous aimerions éviter l'utilisation d'un GPS. De ce fait, nous ne pouvons utiliser ce type de routage indirect.

Dans le cas où le routage est dépendant de la DHT, l'espace virtuel $\mathcal{V}$ qui lui est associé, est utilisé non seulement pour identifier les points de rendez-vous mais également pour router vers ces points et vers la destination finale. Dans ce cas, l'adresse virtuelle d'un

nœud dépend de sa position. La cohérence du protocole de routage repose alors sur la cohérence de la distribution des partitions de l'espace virtuel $\mathcal{V}$ sur les nœuds du réseau. Le routage est effectué sur la structure logique et ne tient plus compte du réseau physique sous-jacent. Dans de tels scenarii, un nœud u cherchant le nœud w récupère l'adresse virtuelle du point de rendez-vous v avec la fonction de hachage $hash(w) = Id_{virtuel}(v)$. Les requêtes de look-up sont routées dans $\mathcal{V}$ jusqu'à v qui retourne l'adresse virtuelle de w. De là, u joint w en utilisant son adresse virtuelle et en routant dans $\mathcal{V}$. Généralement, le routage dans l'espace virtuel est un routage glouton : "Transmet à ton voisin dans $\mathcal{V}$ dont l'adresse virtuelle est la plus proche de l'adresse virtuelle de la destination". C'est ce qu'on trouve par exemple dans Tribe [78, 79] ou L+ [22] sur lequel se base SAFARI [69]. La principale difficulté ici est de répartir les partitions de l'espace virtuel $\mathcal{V}$ de façon à ce que les routes obtenues en routant dans $\mathcal{V}$ ne soient pas beaucoup plus longues que les routes physiques.

5.2 Localisation et routage sur une structure de clusters

Dans notre proposition, chaque nœud détient une information concernant sa position relative : l'identité de son cluster. C'est cette information que les nœuds vont associer à la clef de hachage. Comme nous avons une structure d'arbres, nous proposons de partitionner l'espace virtuel $\mathcal{V}$ dans chaque arbre. $\mathcal{V}$ étant ainsi dupliqué autant de fois que l'on a de clusters, on retrouve un nœud responsable d'une clef donnée dans chaque arbre/cluster. Chaque nœud enregistre alors sa position dans chaque espace virtuel et donc dans chaque arbre. De cette façon, lorsqu'un nœud v recherche un nœud u dans le réseau, il a juste à chercher dans son propre cluster. Comme l'excentricité d'un nœud dans un cluster est faible (voir chapitre 3.6.2), la distance à parcourir pour atteindre le nœud de rendez-vous est également faible. Partitionner ainsi plusieurs fois l'espace virtuel plutôt qu'une seule fois sur tout le réseau évite les situations où la distance entre la source et le point de rendez-vous est supérieure à la distance entre la source et la destination, puisque la source et le point de rendez-vous appartiennent toujours au même cluster alors que la destination peut être n'importe où dans le réseau.

Pour distribuer les partitions de l'espace virtuel $\mathcal{V}$ de la DHT de telle façon que, étant donnée une adresse virtuelle, un nœud u soit en mesure de joindre le nœud en question sans information supplémentaire, nous utilisons un *schéma d'étiquetage d'arbre* (tree Interval Labeling Scheme) pour ensuite permettre un routage par intervalle sur notre structure logique.

Dans les diverses propositions de DHT que nous avons présentées, les deux phases du routage indirect (look-up et routage) sont toujours du même type : indépendantes de la DHT et effectuées dans l'espace physique ou dépendantes de la DHT et effectuées sur l'espace logique. Dans notre approche, les deux étapes de routage indirect sont effectuées différemment. Le look-up est effectué en utilisant un routage par intervalle sur les adresses virtuelles des points de rendez-vous alors que l'étape de routage s'effectue dans l'espace physique, indépendamment de $\mathcal{V}$.

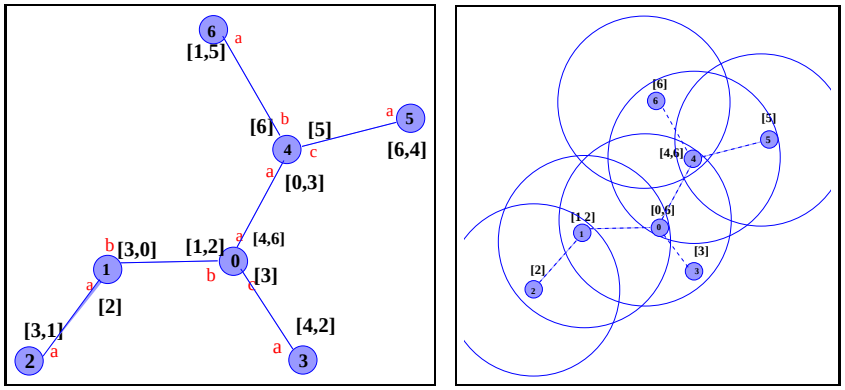
Le routage par intervalle s'est avéré très attractif de par sa simplicité. Il a été introduit dans les réseaux filaires par Santoro et Khatib dans [72] dans le but de réduire la taille des tables de routage. L'idée est de représenter la table de routage de chaque nœud de manière compacte, en agrégeant l'ensemble des adresses destination qui peuvent être atteintes en utilisant le même port de sortie, au moyen d'intervalles d'adresses consécutives. Par exemple, si l'on considère le graphe représenté sur la figure 5.2(a), le nœud 0 doit utiliser le port a pour atteindre les nœuds 4, 5 et 6, le port b pour atteindre 1 et 2 et le port c pour atteindre 3. Plutôt que de créer une entrée dans sa table de routage par nœud destination, il crée une entrée par port de sortie et plutôt que de lister tous les nœuds accessibles via ce port, il stocke seulement l'intervalle contenant les adresses de ces nœuds : $[1, 2]$ pour le port a, $[4, 6]$ pour le port b et $[3]$ pour le port c. Le principal avantage de ce type de routage est qu'il nécessite peu de mémoire sur les nœuds puisque la taille de la table de routage d'un nœud u est en $O(\delta(u))$. Le routage est exécuté de façon distribuée : à chaque nœud intermédiaire x, si x n'est pas le nœud destination y, le message est transféré sur le port de sortie dont l'étiquette est un ensemble d'intervalles I tel que $y \in I$.

Un *Interval Labeling Scheme* (ILS) est la façon d'allouer les étiquettes des nœuds (adresses virtuelles) pour pouvoir définir les intervalles à assigner aux arêtes de chaque nœud, de façon à pouvoir effectuer un routage par intervalle efficace avec des routes aussi courtes que possible. Les auteurs de [77] ont montré qu'un arbre non-orienté supportait un routage par intervalle avec de plus courts chemins (dans l'arbre) et en n'utilisant qu'un intervalle par arête sortante à condition de distribuer les étiquettes (ILS) en effectuant un parcours en profondeur de l'arbre.

Dans les réseaux filaires, dans un arbre, les nœuds doivent stoker un intervalle pour chacune des arêtes sortantes. La taille de la table de routage d'un nœud u est en $O(\delta(u))$. Dans un environnement sans fil, l'émission d'un message atteint tous les nœuds se trouvant à portée radio de l'émetteur. Les arêtes du graphes sont en réalité des hyper-arêtes (voir figure 5.2). La problématique est donc un peu différente puisque une requête émise sera entendue par tous les voisins de la source, qu'elle leur soit destinée ou pas. Comme les nœuds n'ont qu'une hyper-arête sortante, ils peuvent ne stocker qu'un seul intervalle. Nous proposons ici de tirer avantage de la nature diffusante du médium radio. Les nœuds stockent l'intervalle global pour lequel leur sous-arbre est responsable et non plus un intervalle pour chaque voisin. Par exemple, si on considère la figure 5.2, le nœud 0 ne gère plus 3 intervalles mais un seul : $[0, 6]$. Cela donne une taille de table de routage en $O(1)$ par nœud. Quand une requête est envoyée, tous les nœuds à portée radio la reçoivent mais seuls ceux concernés y répondent.

5.2.1 Résumé et analyse de complexité

Pour résumer, nous proposons d'appliquer un routage indirect sur la structure en arbres de notre réseau auto-organisé, en utilisant une DHT qui associe chaque nœud à sa position dans le réseau : son cluster. L'ensemble des adresses virtuelles $\mathcal{V}$ de la DHT est partitionné et réparti sur les nœuds de chaque cluster. Comme le nombre de clusters



(a) Routage en environnement filaire. Les nœuds stockent un intervalle par arête sortante.

(b) Routage en environnement sans fil. Les nœuds stockent un seul intervalle (un par hyper-arête).

FIG. 5.2 – Arêtes dans un réseau filaire (a) Vs hyper-arêtes dans un réseau sans fil (b).

est borné par une constante et que les clusters sont homogènes (chapitre 3), chaque nœud maintient finalement $O(1)$ informations de position.

Quand un nœud u doit joindre un nœud v, il obtient l'adresse virtuelle d'un point de rendez-vous en utilisant la fonction de $hash$: $hash(v) = key_v \in \mathcal{V}$. Puis, à partir de cette adresse, en appliquant un routage par intervalle sur l'espace virtuel $\mathcal{V}$ de son propre arbre, il récupère la position $\mathcal{C}(v)$ de v. Comme les intervalles des voisins de u ne sont pas maintenus par u, u stocke uniquement l'intervalle dont son arbre est responsable. La taille de la table de routage de u est en $O(1)$. Une fois que u connaît $\mathcal{C}(v)$, il peut atteindre $\mathcal{C}(v)$ en employant un routage pro-actif entre clusters puis un routage réactif pour joindre v une fois que le cluster destination est atteint. Comme le nombre de clusters est borné par une constante, chaque cluster a $O(1)$ routes à maintenir vers les autres clusters.

5.3 Notre proposition

5.3.1 Préliminaires

Chaque nœud u possède deux adresses :

– Une adresse universelle $Id(u) \in I\!R$ (souvent notée u par la suite). Cette adresse est unique dans le réseau et ne change jamais. Elle est le "vrai" nom de u.
– Une adresse logique $i(u) \in \mathcal{V}$. Cette adresse est unique dans un cluster. Elle peut changer à chaque re-distribution des partitions de $\mathcal{V}$ parmi les nœuds d'un cluster. Elle identifie le nœud u dans son espace logique.

Soit $I(u)$ la partition de $\mathcal{V}$ assignée au nœud u. $I(u)$ est telle que $I(u) = [i(u), ...[$ où $i(u)$ est l'adresse logique de u dans $\mathcal{V}$. $i(u)$ dépend donc de $I(u)$. On note $I_{tree}(s\mathcal{T}(u)) = \bigcup_{v \in s\mathcal{T}(u)} I(v)$ l'intervalle/partition de $\mathcal{V}$ pour lequel le sous-arbre de racine u est responsable. $|I|$ désigne la taille de l'intervalle I.

5.3.2 Distribution des partitions de l'espace virtuel - ILS

Comme mentionné dans la section 5.2, une distribution optimale des intervalles sur un arbre est obtenue via une numérotation des nœuds obtenue par un parcours en profondeur ILS (*Depth First Search* DFS-ILS).

Comme dans tout ILS traditionnel, les partitions de $\mathcal{V}$ sont attribuées à chaque nœud $u \in V$ de sorte que :
– Les intervalles des nœuds de $s\mathcal{T}(u)$ forment un intervalle continu.
– La taille de l'intervalle dont un sous-arbre est responsable est proportionnelle à la taille de ce sous-arbre : $|I_{tree}(s\mathcal{T}(u))| \propto |s\mathcal{T}(u)|$. Et donc, pour tout nœud $v \in s\mathcal{T}(u)$, $|I_{tree}(s\mathcal{T}(u))| \geq |I_{tree}(s\mathcal{T}(v))|$.
– $\mathcal{V}$ est entièrement distribué parmi les nœuds du cluster : $\mathcal{V} = \bigcup_{v \in \mathcal{C}(u)} I(v)$.
– Les différents intervalles s'excluent mutuellement : $\forall v \in \mathcal{C}(u), \forall w \in \mathcal{C}(u), v \neq w\ I(v) \cap I(w) = \emptyset$.

Nous proposons une répartition des intervalles distribuée, qui s'effectue en parallèle sur chaque branche de chaque arbre. Chaque nœud u nécessite des informations pouvant se trouver jusqu'à $d_{tree}(u, \mathcal{H}(u))$ sauts de lui où $d_{tree}(u, \mathcal{H}(u))$ est le nombre de sauts dans l'arbre entre u et son cluster-head $\mathcal{H}(u)$. La distance $d_{tree}(u, \mathcal{H}(u))$ est bornée par la hauteur des arbres qui est elle-même bornée par une constante (voir chapitre 3.6.2). Cette distribution peut donc être qualifiée de *quasi-locale* selon la taxonomie établie en [81], ce qui implique une maintenance rapide du processus.

Notre algorithme s'exécute en deux temps : une première phase pendant laquelle les nœuds remontent des informations depuis les feuilles de l'arbre jusqu'à sa racine et une seconde phase où les nœuds internes distribuent récursivement les intervalles parmi leurs fils. La distribution des intervalles s'effectue en parallèle sur chaque branche de chaque arbre, chaque phase ayant une complexité temporelle en $O(Tree\_depth)$. La complexité temporelle totale de l'algorithme de distribution des intervalles pour un cluster est donc de $2 \times (Tree\_depth)$. Comme la hauteur des arbres $Tree\_depth$ est bornée par une constante, la complexité temporelle devient $O(1)$.

La figure 5.3 illustre cette distribution d'intervalles pour $\mathcal{V} = [1, 17[$.

Étape 1**.** Comme nous l'avons déjà vu dans le chapitre 3.6.2, chaque nœud u est en mesure de savoir en un temps borné qui l'a choisi comme père parmi ses voisins et donc connaît son nombre de fils. Si un nœud est feuille, la taille de son sous-arbre est 1. Dès qu'un nœud interne a reçu la taille des sous-arbres de chacun de ses fils, il calcule la taille de son propre sous-arbre qui est la somme de la taille des sous-arbres de chacun de ses fils plus 1 : $|s\mathcal{T}(u)| = |\{u\} \bigcup_{v \in \mathcal{C}h(u)} s\mathcal{T}(v)| = 1 + \sum_{v \in \mathcal{C}h(u)} |s\mathcal{T}(v)|$. Chaque nœud envoie la taille de son sous-arbre à son père et ainsi de suite jusqu'à atteindre le cluster-head. C'est ce qu'illustre la figure 5.3(a).

Étape 2**.** Une fois que le cluster-head connaît la taille des sous-arbres de chacun de ses fils, il partage $\mathcal{V}$ équitablement entre lui-même et ses fils. Chaque fils u se voit attribuer une partition de $\mathcal{V}$, $I_{tree}(s\mathcal{T}(v))$, de taille proportionnelle à la taille de son sous-arbre. Chaque nœud interne re-distribue alors l'intervalle qu'on lui a alloué entre lui et ses fils, et ainsi de suite, jusqu'à atteindre les extrémités des branches de l'arbre. Cette étape est illustrée sur la figure 5.3(b).

Ainsi, une fois les deux étapes de l'algorithme accomplies, chaque nœud est responsable d'une partition de $\mathcal{V}$ qui est de taille égale pour tous les nœuds d'un même cluster. La figure 5.3(c) illustre le résultat d'une distribution des intervalles. Le nœud u se voit attribuer l'intervalle $I(u)$ et est désormais responsable des clefs contenues dans cet intervalle et il doit stocker les positions des nœuds v tels que $hash(v) \in I(u)$. On remarque qu'il peut exister plusieurs nœuds v_i tels que $hash(v_i) = hash(v_j)$.

Chaque nœud interne u garde également en mémoire l'intervalle alloué à son sous-arbre $I_{tree}(s\mathcal{T}(u))$, sans pour autant stocker les informations associées à toutes les clefs de $I_{tree}(s\mathcal{T}(u))$. Cela lui servira lors du routage des requêtes, comme nous le verrons dans la section 5.3.6

Comme nous l'avons mentionné dans le chapitre 3.6.2, un nœud interne a en moyenne peu de fils à qui distribuer une partie de $\mathcal{V}$. Cette opération ne génère donc que peu de calcul sur chaque nœud.

5.3.3 Enregistrement

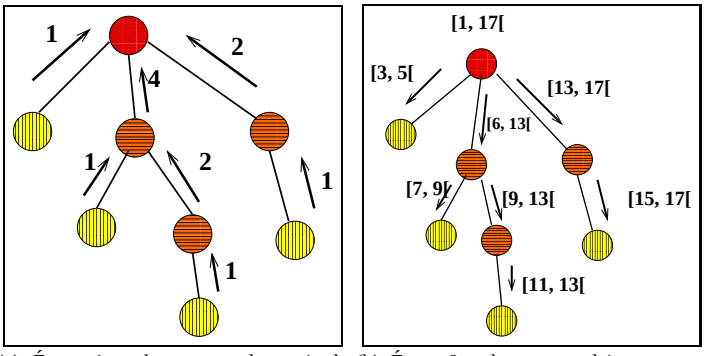
Afin d'être localisé par la suite, chaque nœud u doit enregistrer sa position (l'identité de son cluster) auprès de chaque nœud responsable de la clef $hash(u)$ dans son cluster, mais également dans les autres clusters du réseau. Pour s'enregistrer dans son propre cluster, u a juste besoin d'envoyer une requête d'enregistrement ou *Registration Request*, comme nous le détaillerons plus tard dans la section 5.3.6. Pour s'inscrire dans les autres clusters, u doit tout d'abord joindre un nœud v_i dans chaque cluster $\mathcal{C}_i$. Puis, chaque v_i envoie une *Registration Request* dans son propre cluster $\mathcal{C}_i$ au nom de u. Les nœuds v_i peuvent être trouvés via un routage pro-actif vers le cluster $\mathcal{C}_i$, comme décrit plus tard dans la section 5.3.7.

Les nœuds s'enregistrent toutes les $\Delta(t)$ unités de temps. Une approche communément adoptée dans la littérature [1, 35, 56] est que les informations de localisation sont mises à jour sur les points de rendez-vous en fonction de la distance de ces points de rendez-vous à la source. Plus la distance est courte, plus les mises à jour sont fréquentes.

5.3.4 Départs et arrivées

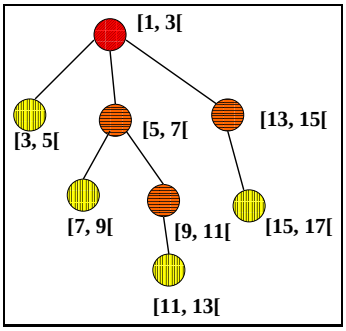
Quand un nœud arrive dans un cluster, il n'est responsable d'aucun intervalle pour un certain temps.

Quand un nœud disparaît, les informations dont il était en charge sont perdues (mais toujours disponibles dans les autres clusters). Chaque nœud interne u est constamment



(a) Étape 1 : chaque nœud envoie la taille de son sous-arbre à son père. Les feuilles (nœuds jaunes hachurés verticalement) envoient 1. Les nœuds internes (nœuds oranges hachurés horizontalement) rassemblent les informations de tous leurs fils et calculent la taille de leur propre sous-arbre avant de l'envoyer à leur propre père, et ainsi de suite, jusqu'à atteindre la racine (le cluster-head), nœud rouge hachuré en diagonal.

(b) Étape 2 : chaque nœud interne partage l'intervalle donné par son père entre lui-même et ses fils, proportionnellement à la taille des sous-arbres de chacun. Les intervalles notés sur les flèches correspondent à ce qui est alloué par un nœud à ses fils, *c.à.d* l'intervalle dont le sous-arbre de chacun est en charge.



(c) Résultat : chaque nœud se voit attribuer un intervalle de $\mathcal{V}$ dont il est responsable.

FIG. 5.3 – Distribution des partitions de l'espace virtuel.

au courant des arrivées et départs de ses fils grâce aux paquets HELLO. S'il constate des changements trop importants parmi eux, il peut décider localement de redistribuer l'intervalle $I_{tree}(s\mathcal{T}(u))$ dont son sous arbre est en charge entre lui-même et ses fils. Plus un nœud interne est proche du chef de cluster, plus les ré-attributions qui découleront de sa décision seront importantes, puisqu'elles se répercutent de père en fils jusqu'à atteindre les feuilles. De plus, une nouvelle attribution des intervalles implique un changement d'adresse logique pour les nœuds. Cependant, comme constaté précédemment, plus un nœud est proche du chef de cluster, plus son voisinage est stable.

Lorsque les intervalles sont re-distribués, chaque nœud u qui était préalablement responsable de l'intervalle I_{old} et qui se voit désormais attribuer la partition I_{new} ne conserve que les informations relatives aux clefs de $I_{old} \cap I_{new}$. De façon à ne pas perdre pour autant les informations associées aux clefs de $I_{old} \setminus I_{new}$, u envoie des *Registration Request* au nom de tous les nœuds v dont il n'est plus en charge *c.à.d* les nœuds v tels que $hash(v) \in I_{old} \setminus I_{new}$.

5.3.5 Ajouter de la redondance et de la robustesse

Comme mentionné dans la section 5.3.4, quand un nœud u disparaît d'un cluster $\mathcal{C}(u)$ ou quitte le réseau, les informations dont il était responsable sont perdues dans ce cluster jusqu'au prochain enregistrement des nœuds. De façon à pallier cet inconvénient, un nœud peut enregistrer sa position d fois dans chaque cluster, d étant une constante. Pour cela, l'espace virtuel $\mathcal{V}$ doit être distribué d fois dans chaque cluster, chaque nœud se voyant désormais attribuer d partitions indépendantes de $\mathcal{V}$ et possédant alors d différentes adresses logiques. Comme d est une constante, la taille mémoire requise sur les nœuds reste en $O(1)$ puisque, en moyenne, si c est le nombre de clusters, un nœud devra stocker $d * c$ positions au lieu de c (c et d étant deux constantes). Ainsi, même lorsqu'un nœud meurt, la position d'un nœud dont il était responsable a toujours des chances d'être trouvée dans le même cluster. Cependant, cela génère plus de messages puisque les requêtes devront désormais suivre d routes, une pour chaque occurrence de $\mathcal{V}$. Dans le pire cas, le routage des requêtes dans l'espace virtuel conduira à une diffusion dans un cluster. Mais, comme nous l'avons étudié dans le chapitre 4, une telle diffusion peut s'effectuer en suivant les branches des arbres de façon efficace et peu coûteuse.

5.3.6 Opération de look-up : Routage dans l'espace virtuel $\mathcal{V}$ de la DHT

Nous détaillons ici comment les requêtes sont routées dans les arbres sur la base d'un routage par intervalle. Ceci correspond à la première étape du routage indirect, l'opération de look-up.

Chaque nœud u a un identifiant unique $Id(u)$. Comme dans tout schéma basé sur une DHT, chaque nœud connaît une fonction spécifique $hash$ qui associe à chaque adresse universelle une adresse logique de l'espace virtuel $\mathcal{V}$.

$$\begin{array}{rll} hash: & I\!R \rightarrow & \mathcal{V} \\ & Id(u) \rightarrow & hash(u) \end{array}$$

Plusieurs nœuds peuvent avoir la même valeur retournée par la fonction $hash$. Les informations de ces nœuds seront stockées sur le même nœud de rendez-vous. Dans la suite, nous utilisons la clef suivante pour identifier un nœud x : $key = \{hash(x), id(x)\}$.

Une requête est routée dans l'espace virtuel jusqu'à atteindre un nœud v responsable de la clef key contenue dans la requête : v est tel que $key \in I(v)$. Une requête peut être de trois types :

- u **veut enregistrer une position :**
  u doit enregistrer sa propre position ou, s'il est un nœud frontière, il peut vouloir enregistrer un nœud u' d'un autre cluster qui lui en a fait la demande. u doit donc trouver le nœud responsable de sa propre adresse logique $hash(u)$ ou de celle de u', $hash(u') : key = \{hash(u), u\}$ (resp. $key' = \{hash(u'), u'\}$)
  Dans ce cas, u utilise une *Registration Request* (RR) $\langle RR, key, \mathcal{C}(u), flag\rangle$ (resp. $\langle RR, key', \mathcal{C}(u'), flag\rangle$).
- u **doit localiser** x **:**
  Dans ce cas, u cherche le nœud responsable de l'adresse logique de x : $hash(x)$.
  Il utilise une *Location Request* (LR) $\langle LR, key = \{hash(x), x\}, i(u), flag\rangle$. L'adresse logique de u, $i(u)$, est ensuite utilisée pour envoyer la requête de réponse à u.
- u **doit répondre à une requête LR concernant une clef dont il est responsable (**$key \in I(u)$**) :**
  u a reçu une requête de type *Location Request* : $\langle LR, key = \{hash(x), x\}, i(w), flag\rangle$ envoyée par le nœud w et contenant une clef telle que $key \in I(u)$. u doit répondre à w.
  Dans ce cas, il utilise une requête de type *Location Reply* (Reply) $\langle Reply, key = \{i(w), -1\}, \mathcal{C}(x), flag\rangle$. On remarque qu'ici la clef key n'a pas besoin de contenir l'adresse réelle de w.

Pour chaque type de requête, le champ $flag$ est mis à 1 par le nœud faisant suivre la requête si la clef key appartient à l'intervalle de son sous-arbre, à 0 sinon. Comme nous le verrons par la suite, ce champ est utilisé pour prendre des décisions de routage.

On remarque qu'un nœud u connaît déjà la position (ou cluster) de ses voisins, de son chef de cluster et des nœuds dont il est responsable. Ainsi, pour un nœud v tel que $\{v \in \mathcal{H}(u) \cup \Gamma_1(u)\}$ ou $hash(v) \in I(u)$, u n'a pas besoin d'effectuer la procédure de look-up et peut directement envoyer le message à v, en suivant la procédure de routage dans le réseau physique, comme décrite dans le chapitre 5.3.7.

Du fait de la nature diffusante du médium radio, un nœud reçoit toutes les requêtes émises par ses voisins, même celles ne le concernant pas. Les requêtes de look-up suivent uniquement les arêtes de l'arbre de *clustering*, donc un nœud ne sera pas

concerné par une requête de look-up provenant d'un de ses voisins qui ne sera ni son père ni l'un de ses fils. Les nœuds prennent la décision de ré-émettre une requête de look-up en se basant non-seulement sur la clé et le champ *flag* contenus dans la requête mais également sur l'identité de leur voisin qui leur l'a transmise. Le champ $flag$ est renseigné à chaque ré-émission de la requête par un nœud. Le processus de routage d'une requête est le même quel que soit son type (LR, RR or Reply). Il est décrit par l'algorithme 1.

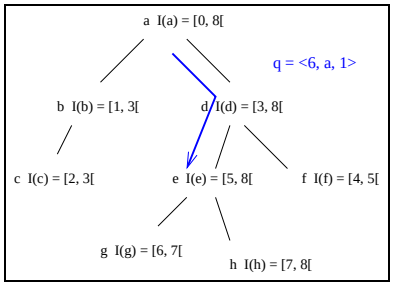
Vu que le routage des requêtes de look-up est opéré sur l'espace virtuel $\mathcal{V}$, il ne nécessite pas la connaissance des adresses universelles des nœuds, il n'a besoin que des adresses logiques.

Sur réception d'un message M (RR, LR or Reply) contenant la clef $key = \{hash(x), x\}$ provenant du nœud u ($u \in \Gamma_1(v)$), le nœud v arrête le processus s'il est responsable de la clef key (donc si $key \in I(v)$) ou s'il est lui-même le nœud cherché (si $key = \{hash(v), v\}$). Ce second cas arrive néanmoins rarement vu qu'il implique que l'initiateur de la requête x soit dans le même cluster que v ($\mathcal{C}(x) = \mathcal{C}(v)$) et que v se trouve sur la route suivie par la requête de x vers le nœud responsable de la clef.

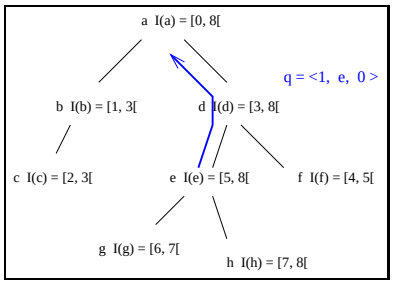
Lorsqu'une requête de type RR $\langle RR, key = \{hash(u), u\}, \mathcal{C}(u), flag\rangle$ atteint sa destination v, v met à jour la position du nœud u dans sa table. Lorsqu'une requête de type Reply $\langle Reply, key = \{i(u), -1\}, \mathcal{C}(x), flag\rangle$ arrive à sa destination u, u est alors capable d'entamer la seconde étape du routage indirect. Enfin, lorsqu'une requête de type LR $\langle LR, key = \{hash(x), x\}, i(w), flag\rangle$ atteint sa destination v (en charge de $hash(x)$), v répond à l'initiateur de la requête par une requête Reply : $\langle Reply, \{i(w), -1\}, \mathcal{C}(x), flag\rangle$.

Si $key \neq \{hash(v), v\}$, le nœud v ré-émet M dans les trois cas suivants :
- Si $u = \mathcal{P}(v)$ et $key \in I_{tree}(s\mathcal{T}(v))$ (M a été transmis à v par son père et le sous-arbre de v contient la clé de la requête). Voir figure 5.4(a).
- Si $u \in \mathcal{C}h(v)$ (M a été transmis à v par un de ses fils u), v ré-émet M si :
  - Si $key \notin I_{tree}(s\mathcal{T}(v))$ (le sous-arbre de v n'est pas responsable de la clef du message), le message doit poursuivre son chemin en remontant dans l'arbre vers la racine. Voir figure 5.4(b).
  - Si $key \in I_{tree}(s\mathcal{T}(v))$ et $key \notin I_{tree}(s\mathcal{T}(u))$ ($flag = 0$) (le sous-arbre de v contient la clef mais pas celui de u), v doit ré-émettre M afin que le message redescende sur une autre branche du sous-arbre de v. Voir figure 5.4(c).

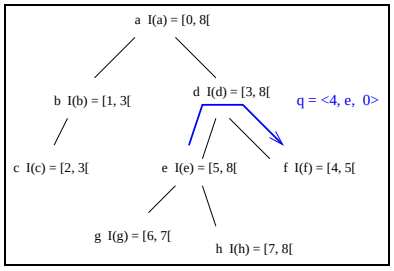
La requête est ignorée dans tous les autres cas, c'est à dire :
- Si $u \notin \mathcal{P}(v) \cup \mathcal{C}h(v)$ (la requête arrive par un lien n'étant pas dans l'arbre). Voir figure 5.5(a).
- Si $u \in \mathcal{C}h(v)$ et $key \in I_{tree}(s\mathcal{T}(u))$ ($flag = 1$) (le message parvient à v par un de ses fils dont le sous-arbre est responsable de la clef). Grâce au champ *flag* mis à 1, u sait que le sous-arbre de l'émetteur est en charge de la clef et donc ne s'en occupe pas. La requête redescend le sous-arbre de v. Voir figure 5.5(b).
- Si $u = \mathcal{P}(v)$ et $key \notin I_{tree}(s\mathcal{T}(v))$ (u reçoit M depuis son père et son sous-arbre n'est pas en charge de la clef). u n'est pas concerné par la requête, il l'ignore. Un de ses frères s'en chargera. Voir figure 5.5(c).



(a) Cas 1 : le message descend dans l'arbre. a cherche le nœud responsable de la clef 6.

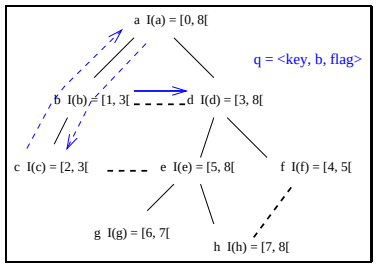


(b) Cas 2 : le message remonte. e cherche le nœud en charge de la clef 1.

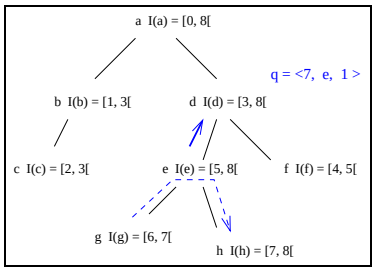


(c) Cas 3 : le message remonte dans une branche pour re-descendre dans une autre. e cherche le nœud en charge de la clef 4.

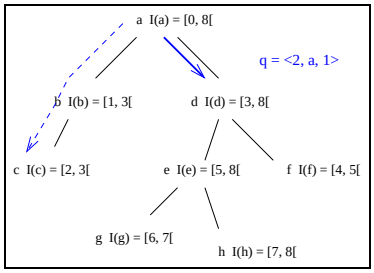
FIG. 5.4 – Différents cas de figures où une requête reçue par d est ré-émise.



(a) Cas 1 : le message ne provient pas d'une branche de l'arbre.



(b) Cas 2 : e cherche le nœud en charge de la clef 4. Le message monte et redescend les branches du sous-arbre de e en étant entendu par d qui ignore la requête.



(c) Cas 3 : a cherche le nœud en charge de 7. Cette clé n'est pas dans l'arbre de nœud d.

FIG. 5.5 – Différents cas de figures où un message est entendu par le nœud d et ignoré. Les flèches pointillées représentent un chemin possible suivi par la requête dans ces cas. Les arêtes pointillées représentent les liens du graphe G non contenus dans l'arbre T.

Algorithm 1 Routage dans l'espace virtuel

Pour tout nœud u, sur réception d'une requête $\langle Type, key = \{hash(x), x\}, X, flag\rangle$, X dépendant du type de requête, provenant d'un nœud $v \in \Gamma_1(u)$ et initiée par le nœud y :

if ($key \in u \cup I(u)$) **then**
▷ *u est responsable de la clef. La requête a atteint sa destination.*
if (Type = LR) **then**
Répond en envoyant $\langle Reply, \{X = i(y), -1\}, \mathcal{C}(x), flag\rangle$. Exit
end
if (Type = RR) **then** Enregistre la position de x. Exit **end**
if (Type = Reply) **then**
Route vers le cluster destination X. Exit
end
end
if ($v = \mathcal{P}(u)$) **then**
▷ *Le message descend les branches de l'arbre.*
if ($key \in I_{tree}(s\mathcal{T}(u))$) **then**
▷ *$\exists w \in s\mathcal{T}(u)$ tel que $key \in I(w)$. cf. figure 5.4(a).*
Met le champ $flag$ à 1.
Ré-émet.
else
Ignore.
▷ *cf. figure 5.5(b).*
end
else
if ($v \in \mathcal{C}h(u)$) **then**
▷ *La requête remonte les branches de l'arbre depuis un fils de u.*
if ($key \notin I_{tree}(s\mathcal{T}(u))$) **then**
Met le champ $flag$ à 0.
Ré-émet.
▷ *cf. figure 5.4(b).*
else
▷ *$\exists w \in s\mathcal{T}(u) \setminus \{u, v\}$ tel que $key \in I(w)$.*
if ($flag = 0$) **then**
Met le champ $flag$ à 1.
Ré-émet.
▷ *$key \notin I_{tree}(s\mathcal{T}(v))$ mais puisque $key \in I_{tree}(s\mathcal{T}(u))$, u doit transmettre la requête à ses autres fils. Celle-ci redescend l'arbre par une autre branche u. Cf. figure 5.4(c).*
else
▷ *La requête transite par v avant de redescendre. Cf. figure 5.5(c).*
Ignore.
end
end
else Ignore.
▷ *Cf. figure 5.5(a).*
end
end

5.3.7 Routage sur le réseau physique

Dans cette section, nous explicitons la seconde phase du routage indirect : le routage dans l'espace physique. Comme nous l'avons déjà mentionné, nous proposons une approche hiérarchique basée sur la structure de clusters dans laquelle nous appliquons un protocole pro-actif entre les clusters (comme par exemple OLSR [25]) et réactif à l'intérieur des clusters (comme par exemple DSR [43] ou AODV [62]). Comme le nombre de clusters est constant quand la densité des nœuds augmente, le nombre de nœuds par cluster augmente aussi ($O(n)$ nœuds par cluster). Comme le nombre de clusters est constant, il en est de même pour le nombre de routes entre ces clusters et chaque cluster a $O(1)$ routes à maintenir vers les autres clusters. Bien que le nombre de nœuds augmente, l'excentricité moyenne des nœuds dans un cluster reste faible et constante (entre 3 et 4 sauts en moyenne). Ainsi, une route réactive dans un cluster peut être trouvée à la demande sans trop de latence et avec un faible nombre de sauts.

Un routage pro-actif entre clusters signifie que chaque cluster ou nœud maintient en permanence la liste des clusters à traverser pour aller d'un cluster A vers un cluster B. Il reste maintenant à définir qui, dans un cluster maintient ces routes pro-actives vers les autres clusters. L'approche la plus communément admise est la suivante. Si très peu de messages sont routés vers les autres clusters, seul le chef de cluster ou quelques nœuds peuvent mémoriser ces routes et les fournir sur demande aux autres nœuds du cluster. Si au contraire, cela arrive plus fréquemment, la table de routage entre clusters peut être distribuée à tous les nœuds en appliquant par exemple l'algorithme de diffusion efficace dans un cluster introduit en section 4.4.

Supposons que le nœud u cherche à joindre le nœud v. Si v est le chef de cluster de u ou un de ses voisins ($v \in \Gamma_1(u) \cup \{\mathcal{H}(u)\}$), u sait déjà comment joindre v et n'a donc pas besoin de faire appel à la fonction de look-up. Dans le cas contraire, u doit d'abord localiser v (connaître $\mathcal{C}(v)$) avant de pouvoir ensuite lui envoyer un message suivant le processus de routage illustré sur la figure 5.6. Si $\mathcal{C}(v) = \mathcal{C}(u)$, alors u initialise un routage de type réactif dans son cluster pour joindre v. Sinon, grâce au routage pro-actif entre clusters, u connaît la liste des clusters à traverser pour atteindre $\mathcal{C}(v)$. Soit $\mathcal{C}(w)$ le premier cluster à traverser. u initialise un routage réactif vers un nœud $x \in \mathcal{C}(u)$ qui est un nœud frontière avec $\mathcal{C}(w)$: x tel que $x \in \mathcal{C}(u)$ et $\exists y \in \Gamma_1(x) \cap \bar{\mathcal{C}}(w)$.

x fait alors suivre le message de u à l'un de ses voisins y se trouvant dans $\mathcal{C}(w)$. y réitère alors le même processus de routage, et ainsi de suite, jusqu'à joindre le cluster de la destination $\mathcal{C}(v)$.

Ce processus de routage est donné dans l'algorithme 2. Nous notons $Next\_Hop(cluster_1, cluster_2)$ la fonction qui retourne le prochain cluster à traverser pour atteindre le $cluster_2$ depuis le $cluster_1$. Cette fonction illustre le routage pro-actif entre clusters et est connue par tous les nœuds.

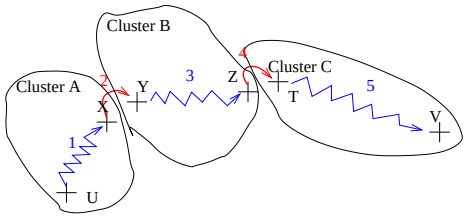


FIG. 5.6 – u souhaite joindre le nœud v. Grâce à l'opération de look-up, il sait que $\mathcal{C}(v) = C$. Il connaît le prochain cluster à traverser pour atteindre C : le cluster B. Il joint le nœud x, voisin de B avec un protocole réactif (flèche 1). x transmet le message à son voisin y du cluster B (flèche 2). y sait que les clusters B et C sont voisins, il transmet le message à un nœud z de son cluster, voisin du cluster C (flèche 3). z transmet à son voisin t dans C (flèche 4) qui joint finalement v grâce à un routage réactif dans son cluster (flèche 5).

Algorithm 2 Routage hiérarchique

Pour un message M **envoyé par le nœud** $x \in \mathcal{C}(x)$ **au nœud** $y \in \mathcal{C}(y)$
  $\mathcal{C}_{current} = \mathcal{C}(x)$
  $\mathcal{C}_{next} = \mathcal{C}(x)$
  while ($\mathcal{C}_{next} \neq \mathcal{C}(y)$) do
    $\mathcal{C}_{next} = Next\_Hop(\mathcal{C}_{current}, \mathcal{C}(y))$
    Envoie M avec un routage réactif vers le nœud $u \in \mathcal{C}_{current}$ tel que $\exists v \in \Gamma_1(u) \cap \mathcal{C}_{next}$.
    u envoie M à son voisin v.
    $\mathcal{C}_{current} = \mathcal{C}_{next}$
  end
  ▷ *Le message a atteint le cluster destination.*
  Envoie M avec un routage réactif vers le nœud destination y.
end

Enregistrer la position d'un nœud . Si le nœud u souhaite seulement enregistrer sa position dans un autre cluster que le sien, il utilise le même schéma. Par exemple, si u souhaite s'enregistrer dans C, il exécute l'algorithme 2 (sans avoir à effectuer l'opération de look-up vu qu'il connaît déjà le nom des clusters existants ainsi que la liste des clusters à traverser pour les joindre) jusqu'à atteindre un nœud dans C, quel qu'il soit (le nœud t dans notre exemple). t envoie alors une *Registration Request :* $\langle RR, key = \{hash(u), u\}, A, flag\rangle$ dans C. On remarque que, comme la requête passe par le nœud y dans le cluster B, y peut faire de même et enregistrer u dans b en même temps.

5.4 Simulations

Comme nous l'avons déjà mentionné, il n'a été proposé, à notre connaissance, qu'un autre protocole de routage hiérarchique qui propose une approche réactive à l'intérieur des clusters et pro-active entre les clusters : le protocole SAFARI [69]. Dans cette section, nous évaluons par simulation notre algorithme de localisation/routage sur notre structure de clusters en la comparant aux performances de SAFARI.

5.4.1 SAFARI

SAFARI propose une organisation hiérarchique de l niveaux, *c.à.d* que les clusters (appelés *cellules* dans SAFARI) sont récursivement re-groupés en clusters de niveaux supérieurs et ainsi de suite. Les simples nœuds sont considérés comme des clusters/cellules de niveau 0. Le nombre de niveaux s'établit automatiquement en fonction de la taille et de la densité du réseau.

Le rayon des cellules de SAFARI est défini *a priori*. Le rayon D_1 des cellules de niveau 1 (équivalentes aux clusters de notre algorithme, également appelées cellules fondamentales) est fixé et les rayons D_i des niveaux supérieurs sont définis récursivement à partir de D_1. La hiérarchie de cellules construite par SAFARI se base sur une auto-sélection des nœuds en tant que *drums* (cluster-heads). Un drum de niveau i est également un drum de tout niveau inférieur j tel que $0 \leq j \leq i$. Un drum de niveau i décide d'augmenter ou de décrémenter son niveau en fonction du nombre de drums de niveaux $i+1$ et i se trouvant à une certaine distance de lui. Si un drum de niveau i n'a aucun drum de niveau $i+1$ parmi ses voisins à moins de D_i sauts, il incrémente son niveau et s'auto-déclare drum de niveau $i+1$. Si deux drums de niveau i sont distants de moins de D_i sauts, seul le drum de plus grand identifiant reste drum de niveau i, l'autre décrémente son niveau et devient drum de niveau $i-1$.

Cette hiérarchie attribue à chaque nœud un ancêtre (chef) unique à chaque niveau. A partir de cette algorithme de clustering hiérarchique, chaque nœud se voit attribuer une adresse/coordonnée logique (qui sera son adresse dans l'espace de la DHT). La coordonnée d'un nœud de niveau i est la concaténation de la coordonnée de son drum de niveau $i+1$ et d'un nombre généré aléatoirement.

Soient $COORD(d_i)$ la coordonnée d'un drum d_i de niveau i, $PARENT(d_i)$ le père de d_i (le drum de niveau $i+1$ de d_i) et $Rand$ un nombre aléatoire. La coordonnée de d_i est comme suit :

$$\begin{array}{lcll} COORD(d_i) & = & COORD(PARENT(d_0)) & \text{pour } i = 0 \\ & = & COORD(PARENT(d_i)).Rand & \text{pour } 0 < i \end{array}$$

Les drums de niveau 0 (les simples nœuds) sont des feuilles dans cet arbre de coordonnées. Tous les nœuds d'une même cellule fondamentale ont la même coordonnée logique.

Chaque drum de niveau i envoie un paquet appelé *beacon* toutes les T_i unités de temps, T_i dépendant du niveau i. Plus le niveau hiérarchique est élevé, plus la période

d'émission des *beacons* correspondants est grande. Un *beacon* de niveau i envoyé par le drum d_i est transmis à tous les nœuds de la cellule de d_i ainsi qu'à tous les nœuds se trouvant dans une cellule de niveau i dont le drum de niveau $i+1$ est le père de d_i. Par exemple, sur la Figure 5.4.3, les *beacons* de niveau 1 envoyés par le drum de niveau 1 de la cellule F seront envoyés à tous les nœuds des cellules F et G, puisque ces deux cellules appartiennent à la même cellule de niveau 2. Chaque nœud stocke tous les *beacons* qu'ils relaient dans une table appelée Drum Ad Hoc Routing Table (DART) en leur associant la date de réception et le nœud par lequel il a été reçu.

Les coordonnées des nœuds forment l'espace d'adressage de la DHT de SAFARI. La fonction de hachage retourne k différentes coordonnées de points de rendez-vous pour chaque niveau i. Contrairement à notre heuristique où les points de rendez-vous se trouvent dans le même cluster que le demandeur, les points de rendez-vous dans SAFARI sont répartis sur l'ensemble du réseau. Le look-up se base sur l'idée qu'en général, les nœuds communiquent plus avec les entités proches d'eux. Lorsqu'un nœud x veut s'enregistrer, il hache son identifiant et obtient k coordonnées pour chaque niveau i ($0 \leq i \leq l$) par la DHT. x va s'inscrire k fois dans chaque niveau. Pour chacune de ces coordonnées c retournée par la DHT, il va s'enregistrer auprès du nœud qu'il trouve qui a la coordonnée la plus proche possible de la coordonnée c. Pour cela, il envoie sa requête d'enregistrement au nœud u se trouvant dans sa DART et dont la coordonnée est la plus proche de la coordonnée c. u fait de même et transmet la requête de u au nœud de sa propre DART dont la coordonnée est la plus proche de c et ainsi de suite jusqu'à atteindre un nœud d'une cellule fondamentale (tous les nœuds de la même cellule fondamentale ont la même coordonnée) qui n'a aucune entrée dans sa DART avec une coordonnée plus proche de c qu'il ne l'est lui-même. Ce nœud [3] devient le nœud auprès duquel le nœud u enregistre sa position. Lorsqu'un nœud x veut récupérer la coordonnée d'un nœud y, il va d'abord chercher dans les cellules de niveau 1 appartenant à la même cellule de niveau 2 que lui. S'il ne trouve pas, il cherchera au niveau supérieur et ainsi de suite : il cherche dans toutes les cellules de niveau i appartenant à la même cellule de niveau $i+1$ que lui, jusqu'à atteindre l'ensemble du réseau. Lorsqu'un nœud x veut récupérer les coordonnées d'un nœud y, il hache la coordonnée de y et envoie sa requête au nœud u se trouvant dans sa DART et dont la coordonnée est la plus proche de la coordonnée retournée par la DHT pour le niveau considéré, et ainsi de suite, jusqu'à atteindre un nœud r. Si r connaît les coordonnées de y, il les retourne à x qui pourra alors joindre y de la même façon qu'il a joint r. Si r ne détient pas la coordonnée de y, x ré-itère sa requête au niveau supérieur. x peut donc envoyer jusqu'à l requêtes de look-up pour localiser y. Nous verrons par la suite que même au bout de l fois, le look-up de SAFARI peut échouer. Dans nos simulations, k est fixé à 3, comme le suggère les auteurs de SAFARI.

Nous nous sommes intéressés dans un premier temps à la formation des clusters de chacun des deux algorithmes, puis, nous avons évalué les performances du look-up et du routage de chacun des protocoles.

[3] Tous les nœuds d'une cellule fondamentale ayant la même coordonnée, plusieurs nœuds peuvent potentiellement stocker la position de u mais SAFARI n'explicite pas s'il s'agit du drum ou d'un nœud particulier de la cellule

5.4.2 Comparaison des structures

Comme notre heuristique construit des clusters de rayon compris entre 3 et 4 sauts (chapitre 3), nous avons fixé D_1 à 3 dans SAFARI, de façon à pouvoir comparer les cellules fondamentales de SAFARI aux clusters de notre algorithme. C'est d'ailleurs la valeur choisie par les auteurs de SAFARI dans leurs simulations.

La différence principale entre les deux heuristiques est que SAFARI construit l niveaux hiérarchiques de cellules alors que notre algorithme ne forme qu'un seul niveau de clusters. Le nombre de niveaux établi par SAFARI s'adapte automatiquement en fonction de diamètre du réseau. La table 5.1 donne le nombre de niveaux construits par SAFARI sur différentes topologies. Nous verrons par la suite que le nombre de niveaux impacte les performances du look-up de SAFARI.

Topologie	Nombre de niveaux
10×10 Grille à 4 voisins	entre 3 et 4
15×15 Grille à 4 voisins	4
Chaîne de 50 nœuds	entre 4 et 5
Chaîne de 75 nœuds	5
Chaîne de 100 nœuds	6
Topologie Poisson $\lambda = 500$, $R \leq .1$	entre 3 et 4
Topologie Poisson $\lambda = 500$, $R < .1$	entre 2 et 3

TAB. 5.1 – Nombre de niveaux de cellules construits par SAFARI en fonction de la topologie sous-jacente.

Bien que le diamètre du réseau intervienne sur le nombre de niveaux hiérarchiques, seul le degré des nœuds influence les caractéristiques des clusters, tout comme dans notre algorithme, les heuristiques étant locales et distribuées. Les résultats de la table 5.2 montrent que les clusters construits par les deux heuristiques ont des caractéristiques moyennes équivalentes (nous ne considérons que les clusters de niveau 1 pour SAFARI). Cependant, comme le montre l'écart type du nombre de nœuds par cluster, les clusters de SAFARI sont moins homogènes que ceux de notre heuristique.

Comme nous l'avons étudié dans le chapitre 3, lorsqu'un nœud intègre le réseau organisé avec notre algorithme, il vérifie son voisinage, calcule sa densité et se choisit un père. L'algorithme stabilise rapidement en un temps proportionnel à la hauteur de l'arbre. Dans SAFARI, lors de la phase d'initialisation, les nœuds attendent un temps aléatoire avant de prendre la décision d'éventuellement augmenter leur niveau. Cette décision est basée sur les informations contenues dans la DART de chaque nœud. Le temps de stabilisation de SAFARI est donc lié à la période initiale d'attente aléatoire et à la fréquence T_i d'émission des *beacons* (donc à T_1 pour les cellules fondamentales). Dans nos simulations, les nœuds tirent un temps aléatoire uniformément entre 0 et 5 unités de temps et T_1 = 2 unités de temps, comme le suggèrent les auteurs de SAFARI.

De façon à comparer équitablement les deux heuristiques, nous supposons que les *beacons* de niveau 1 sont échangés à la même fréquence que les paquets Hello dans notre

heuristique (T_1).

λ	500		600		700	
	Densité	SAFARI	Densité	SAFARI	Densité	SAFARI
Nb clusters	11.70	16.2	10.08	12.6	8.06	11.4
Taille clusters	39.91	32.58	45.64	39.76	54.43	43.57
$\sigma(taille)$	18.66	13.88	17.88	17.83	16.59	20.28
Diamètre	4.99	4.67	5.52	4.62	5.50	4.76
CH/drum excen.	3.01	2.69	3.09	2.67	3.37	2.80
Temps stab.	5.27	107.67	5.34	113.41	5.33	91.95
$\sigma(Temps\ stab)$	0.63	132.41	0.74	135.56	0.85	123.69
λ	800		900		1000	
	Densité	SAFARI	Densité	SAFARI	Densité	SAFARI
Nb clusters	7.03	9.10	6.15	8.10	5.57	7.40
Taille clusters	61.23	54.80	70.41	60.58	73.72	66.21
$\sigma(taille)$	15.59	23.20	15.29	25.01	14.27	26.61
Diamètre	5.65	4.83	6.34	4.77	6.1	4.73
CH/drum excen.	3.17	2.77	3.19	2.72	3.23	2.82
Temps stab.	5.34	90.55	5.43	60.61	5.51	61.97
$\sigma(Temps\ stab)$	0.99	111.18	1.21	115.58	1.44	118.69

TAB. 5.2 – Caractéristiques des clusters pour chaque heuristique pour $R = .1$.

Le temps de stabilisation des algorithmes est présenté dans la table 5.2. Parfois, nous avons pu remarquer qu'au bout de 350 unités de temps (temps suggéré par les auteurs de SAFARI), la structure de SAFARI n'était pas établie. Dans ces cas, SAFARI ne converge pas. Nous n'avons pris en compte dans ces statistiques que les cas où SAFARI converge. On remarque que l'intensité des nœuds n'influence pas le temps de stabilisation des deux protocoles. SAFARI est beaucoup plus long à stabiliser que notre algorithme et son temps de stabilisation est loin d'être régulier comme le montre l'écart type σ. Certaines instances du protocole convergent très rapidement alors que d'autres n'ont toujours pas convergé après un temps de 350 unités de temps. En effet, un nœud peut osciller entre différents états en fonction des valeurs aléatoires choisies.

La figure 5.7 donne un exemple dans lequel SAFARI ne converge pas. Dans la figure 5.7(b), le réseau est en phase d'initialisation. La période d'attente aléatoire du nœud 3 est la première à expirer : le nœud 3 devient le premier drum de niveau 1. Puis, les périodes d'attente de plusieurs nœuds expirent. Ceux parmi eux qui entendent un drum de niveau 1 à moins de $D_1 = 3$ sauts s'attachent à lui. Les autres s'auto-élisent drum de niveau 1. Dans notre exemple, la période du nœud 1 a expiré avant celles de 4, 12 et 25 et celle de 6 a expiré avant celles des nœuds 23, 8, 26 et 16. Si au contraire, la période du nœud 23 avait expiré avant celle du nœud 6, par exemple, 23 aurait créé son propre cluster et 6 se serait ensuite rattaché à 23. D'un autre côté, on peut remarquer que si la période du nœud 19 expire avant celle du nœud 1, 19 s'attache dans un premier temps au drum 3 avant de s'attacher à 1 lorsque celui-ci se déclare drum. Ainsi, sur la

figure 5.7(c), les nœuds 6 et 1 deviennent des drums de niveau 1, les nœuds 0, 9, 7 s'attachent au nœud 0, les nœuds 21, 19, 12, 4 et 25 s'attachent au nœud 1 et les nœuds 0, 7 et 9 s'attachent à 3. Ceci montre l'importance du temps d'attente aléatoire dans la formation des clusters de SAFARI. Comme il n'existe aucun drum de niveau 2 à moins de $D_2 = 6$ sauts du nœud 3, celui-ci devient ensuite un drum de niveau 2. les cellules de niveau 1 des drums 1 et 6 s'attachent à la cellule de niveau 2 de 3 (figure 5.7(d)). Puis, la période d'attente du nœud 17 expire. Comme celui-ci n'entend aucun drum de niveau 1 à moins de D_1 sauts, il devient un drum de niveau 1. Lorsque les autres nœuds se réveillent, ils s'attachent à lui. Comme le drum 17 de niveau 1 n'entend aucun drum de niveau 2 dans son voisinage à D_2 sauts, il devient un drum de niveau 2. Sa cellule de niveau 2 ne se compose que d'une seule cellule de niveau 1. De la même façon, comme le drum 3 de niveau 2 n'entend aucun drum de niveau 3 dans son voisinage à D_3 sauts, il devient un drum de niveau 3. C'est ce que l'on peut voir sur la figure 5.7(f). Dans SAFARI, si un drum de niveau i n'entend pas au moins deux drums de niveaux $i-1$, il baisse son niveau. Ainsi, dans notre exemple, comme le drum 3 de niveau 3 n'entend qu'un seul drum de niveau 2 (le drum 17), il re-devient un drum de niveau 2. De même, comme le drum 17 de niveau 2 n'entend aucun drum de niveau 1, il re-devient drum de niveau 1. C'est à dire que nous revenons à la structure de la figure 5.7(e). La structure va ensuite évoluer pour re-devenir celle illustrée par la figure 5.7(f) et ainsi de suite. Un cycle apparaît et SAFARI ne converge jamais.

5.4.3 Look-up et routage

Cette section analyse les performances du look-up effectué dans chaque algorithme ainsi que le routage qui s'en suit.

Dans notre algorithme, les requêtes de look-up sont routées à l'intérieur d'un cluster en effectuant un routage par intervalle sur l'espace virtuel de la DHT (Section 5.3). Les diamètres des clusters étant relativement petits, la requête ne parcourra qu'un nombre de sauts borné pour atteindre le nœud rendez-vous. Ceci n'est pas le cas dans SAFARI où les nœuds rendez-vous se trouvent dans la plupart des cas dans d'autres clusters. De plus, comme nous l'avons déjà évoqué, dans un cas de réseau statique, avec une couche MAC idéale, tous les look-ups de notre algorithme réussiront, ce qui n'est pas le cas dans SAFARI. Et même lorsqu'un look-up de SAFARI réussit, il peut avoir utilisé plusieurs requêtes alors qu'une seule suffit à notre algorithme. Cela s'explique par le phénomène suivant. Les drums de niveau i de SAFARI envoient leurs *beacons* aux nœuds des cellules de niveau i ayant le même drum $i+1$ qu'eux. Donc, dans une hiérarchie de 3 niveaux ou plus, tous les nœuds ne recevront pas les *beacons* de tous les drums. Si on prend l'exemple de la figure 5.4.3, le drum de la cellule fondamentale B envoie ses *beacons* aux nœuds des cellules A, B et C mais les nœuds des cellules D ou E ne les reçoivent pas.

Quand le nœud d cherche à s'enregistrer, il hache son identifiant et regarde dans sa table DART quel nœud v a la coordonnée la plus proche de la valeur retournée par la DHT. En effectuant le look-up sur son propre identifiant, d finit par s'enregistrer auprès d'un nœud h. Cependant, il faut noter que h est le nœud trouvé à partir de la DART

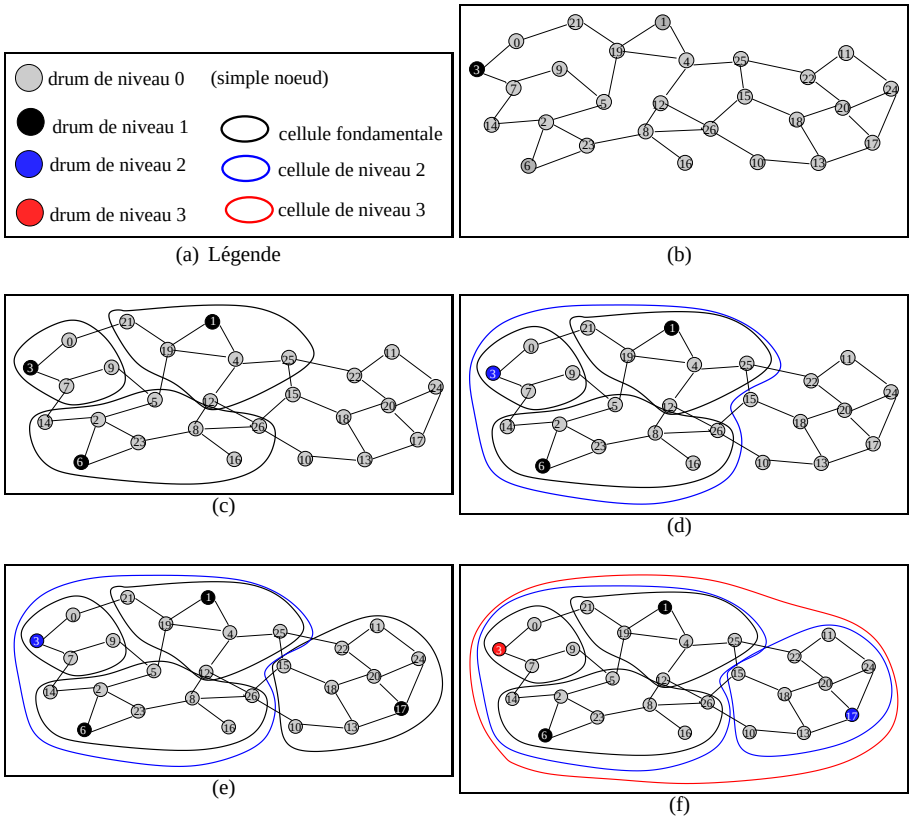


FIG. 5.7 – Exemple où SAFARI oscille et ne converge jamais.

de d. h n'a pas toujours exactement la même coordonnée que celle retournée par la DHT, il est juste un nœud dont la coordonnée est proche. Le problème vient alors du fait que les nœuds possèdent une table DART différente les uns des autres lorsque le cluster est organisé en plus de 2 niveaux hiérarchiques. En effet, quand un nœud s veut envoyer un message à d, il hache l'identifiant de d et ainsi obtient des coordonnées. Il va alors envoyer sa requête aux nœuds de sa DART dont les coordonnées sont les plus proches de celles retournées par la DHT. Il va atteindre un nœud n. Or d ne s'est pas enregistré auprès de n car n de figure pas dans la DART de d et n'est pas atteignable depuis la DART de d. De même, les nœuds h auprès desquels d s'est enregistré ne sont pas toujours contenus dans la table DART de s et dans ces cas-là, le look-up, ne peut réussir. Ainsi, plus le nombre de niveaux hiérarchiques est élevé, plus le taux de réussite des look-ups de SAFARI est faible.

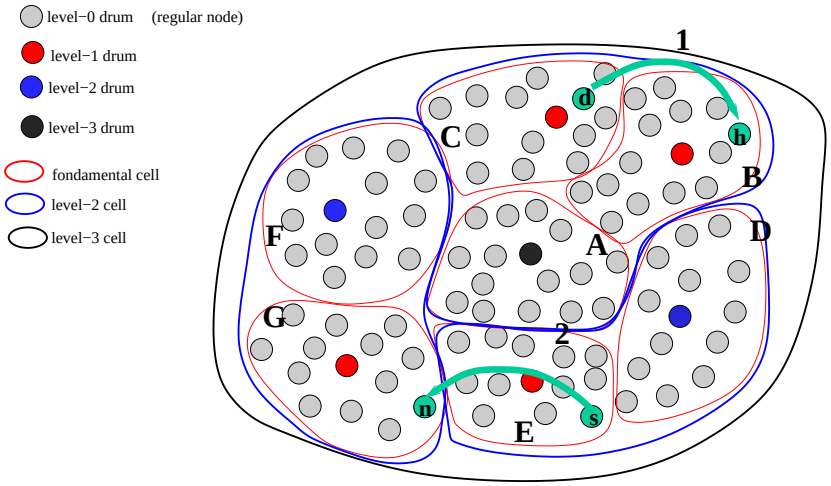


FIG. 5.8 – Exemple de clusters de SAFARI.

La table 5.3 présente différentes valeurs que nous avons relevées lors de nos simulations de look-up et routage de chacun des deux algorithmes. Là encore, les valeurs concernant SAFARI ne sont prises en compte que lorsque l'algorithme converge.

Le champ "Nb de requêtes" indique le nombre de requêtes qu'un nœud u doit envoyer en moyenne avant de joindre un nœud qui détient l'information recherchée (avant de réussir le look-up). Ce champ est toujours égal à 1 pour notre heuristique puisqu'une seule requête est nécessaire.

Le champ "Longueur du Look-up" donne le nombre de sauts que parcourt une requête de look-up en moyenne dans chaque algorithme. Les nœuds de rendez-vous de notre algorithme étant toujours dans le même cluster que la source de la requête et ceux de SAFARI étant distribués sur l'ensemble du réseau, les routes sont évidemment plus courtes dans notre heuristique. À cela s'ajoute que SAFARI peut lancer une requête de plus en plus loin en augmentant le niveau après un échec de requête de look-up. Ces valeurs sont influencées par la densité locale du réseau (degré des nœuds) puisque les

λ	500		600		700	
	Densité	SAFARI	Densité	SAFARI	Densité	SAFARI
Nb de requêtes	1	1.71	1	1.82	1	1.78
Taux de réussite	100%	95.70%	100%	92.20%	100%	90.90%
Longueur du Look-up	3.02	14.94	3.07	12.56	3.15	10.68
Longueur des chemins	6.31	7.28	6.67	5.87	6.37	6.17
Longueur globale	12.39	37.16	12.81	30.99	12.67	27.53
λ	800		900		1000	
	Densité	SAFARI	Densité	SAFARI	Densité	SAFARI
Nb de requêtes	1	1.79	1	1.58	1	1.54
Taux de réussite	100%	85.80%	100%	90.50%	100%	91.00%
Longueur du Look-up	3.16	10.36	3.21	8.63	3.24	5.04
Longueur des chemins	6.75	5.88	6.61	5.73	6.66	5.09
Longueur globale	13.07	26.60	13.03	22.99	13.14	15.17

TAB. 5.3 – Comparaison de notre algorithme et de SAFARI.

caractéristiques des clusters en découlent. Cependant, ce facteur dépend également de l'étalement du réseau pour SAFARI car plus le réseau est étendu, plus le nœud rendez-vous peut être éloigné.

Le champ "Longueur des chemins" donne le nombre de sauts à parcourir dans la deuxième phase du routage indirect, en suivant les schémas de routage proposés dans chacun des algorithmes. Le champ "Longueur globale" donne le nombre moyen de sauts à parcourir avant d'atteindre enfin le nœud destination. Il est égal à la longueur des routes de la deuxième étape plus deux fois la longueur des routes du look-up car la requête de look-up doit effectuer un aller-retour.

On remarque que les chemins de look-up dans SAFARI sont plus longs que les chemins empruntés par les messages lors de la deuxième phase du routage indirect, ce qui peut induire une latence importante. Ce n'est pas le cas dans notre approche où les requêtes de look-up sont contenues dans un cluster et donc suivent un chemin dont la taille est bornée par le diamètre des clusters, lui-même borné par une constante. (Cf. chapitre 3).

Les longueurs des routes (suivies par les requêtes de look-up et par les messages de données) dépendent bien sûr de la densité locale du réseau comme toutes les autres caractéristiques étudiées pour ces deux protocoles. Cependant, les routes empruntées dans la deuxième phase du routage indirect dépendent également de l'étalement du réseau. Plus le réseau est grand, plus la distance séparant deux nœuds du réseau est grande. Dans SAFARI, il en est de même pour la taille des routes suivies par les requêtes de look-up. Dans notre approche, la longueur des routes du look-up reste constante lorsque le réseau s'étale car elle est limitée par le diamètre des clusters qui lui-même reste constant. Ainsi, même si pour les résultats de simulation obtenus, la longueur des routes de look-up dans notre approche représente près de la moitié de la taille du chemin global, elle tend à devenir négligeable devant la taille des routes vers le nœud destination lorsque le réseau s'étale, ce qui n'est pas le cas de SAFARI pour lequel le ratio $\frac{longueur\ look-up}{longueur\ totale}$ reste constant avec l'étalement du réseau. Les

comparaisons menées ici peuvent sembler avoir été menées sur un réseau trop peu étendu. Cependant, le but recherché ici était de comparer notre algorithme à SAFARI et pour un réseau plus large, SAFARI construit un plus grand nombre de niveaux et ne converge plus. C'est pourquoi, nous n'avons pu simuler le routage sur un réseau plus étendu qu'avec notre métrique. Les résultats donnés dans la table 5.4 sont obtenus sur un réseau où le degré moyen des nœuds (et donc l'intensité λ du processus de Poisson) est constant mais avec une taille de réseau de plus en plus importante. Les résultats sont donnés pour $\lambda = 500$ ($\overline{\delta} \approx 15.7$) mais le comportement de l'algorithme est le même quelle que soit l'intensité considérée.

Les résultats montrent qu'effectivement, bien que le réseau grandisse, la longueur des routes empruntées par les requêtes de look-up reste constante et que seule la longueur totale des routes pour joindre le nœud destinataire augmente.

Les routes empruntées lors de la deuxième phase du routage indirect ne sont pas optimales dans la mesure où elles ne sont pas calculées sur la topologie des nœuds mais sur la topologie de clusters. On retrouve le même principe que dans BGP [4] où les séquences des AS (Autonomous System) ne donnent pas toujours le nombre de sauts optimal entre clusters. Le champ "Étirement" donne l'écarte en nombre de sauts entre la longueur des routes empruntées lors de la deuxième phase du routage indirect et la longueur des routes dans le graphe (plus courts chemins) en nombre de sauts. On remarque que le facteur d'étirement survenant dans la seconde phase du routage indirect est négligeable devant la longueur totale des routes. Seul l'ajout des sauts nécessaires au look-up importe.

Nb de nœuds	500	600	700	800	900	1000
Nb de Clusters	11.70	14.20	15.80	17.50	21.42	24.30
Longueur du Look-up	3.02	2.97	3.07	3.05	2.99	3.07
Longueur des chemins	6.31	6.88	7.08	8.06	8.69	9.05
Étirement	0.69	0.74	0.78	0.82	0.86	0.92
Longueur globale	12.39	12.90	13.55	13.97	14.98	15.43

TAB. 5.4 – Propriétés de notre approche de routage avec $\lambda = 500$ (degré moyen des nœuds constant $\overline{\delta} = 15.7$) lorsque le réseau s'étale.

5.5 Conclusion

Dans ce chapitre, nous avons proposé un protocole de routage hiérarchique pouvant s'appliquer sur notre organisation de cluster. Cette approche hiérarchique suit le schéma inverse que ceux communément décrits dans la littérature. En effet, nous proposons d'appliquer un protocole de routage réactif à l'intérieur des clusters et pro-actif entre les clusters. Une telle approche suppose un routage indirect utilisant une table de

[4] www.freesoft.org/CIE/RFC/1772/

hachage distribuée. Notre approche tire avantage des caractéristiques intrinsèques du médium radio et des clusters pour à la fois proposer un schéma d'étiquetage efficace des sommets des arbres de *clustering* pour distribuer les partitions de l'espace logique d'adressage de la DHT et permettre un routage global impliquant une taille mémoire sur les nœuds en $O(1)$. Les requêtes de look-up peuvent ensuite être routées grâce à un routage par intervalle sur cet espace. La deuxième phase du routage indirect se fait alors dans l'espace physique avec des chemins quasi-optimaux.

Afin d'évaluer les performances de notre algorithme, nous l'avons comparé au seul autre protocole de notre connaissance qui utilise le même schéma de routage hiérarchique : SAFARI. Il s'est avéré que notre approche offre de meilleures caractéristiques sur différents critères : temps de stabilisation, succès des look-ups, taille des chemins, etc. Cependant, ces résultats sont à modérer de par le fait qu'il n'existe encore que très peu de protocoles proposant cette approche de routage hiérarchique.

Dans la continuité de cette approche, nous souhaiterions analyser plus en profondeur les périodes de rafraîchissement des enregistrements des nœuds, en fonction de leur mobilité. Comme les algorithmes présentés dans cette thèse peuvent être qualifiés de "quasi-locaux" et que la structure de clusters sous-jacente a présenté de bons comportements face à la mobilité des nœuds, nous espérons qu'il en sera de même pour le protocole de localisation/routage.

Comme nous l'avons vu, seulement peu de propositions utilisent une approche proactive entre les clusters et réactive à l'intérieur des clusters. Nous avons comparé notre approche à une autre qui utilisait ce même modèle. Cependant, il peut également s'avérer intéressant d'établir des comparaisons avec une approche "classique" de la littérature, *c.à.d.* qui propose un routage réactive entre clusters et pro-actif au sein d'un cluster.

5.6 Publications

1. **Journaux et revues avec comité de lecture :**
   (a) *Distributed Node Location in clustered multi-hop wireless networks.* Nathalie Mitton et Éric Fleury. GESTS International Transaction on Computer Science and Engineering, Volume 21, Décembre 2005.
2. **Colloques et conférences internationaux avec comité de lecture :**
   (a) *Distributed Node Location in clustered multi-hop wireless networks.* Nathalie Mitton, Éric Fleury. Asian Internet Engineering Conference (AINTEC'05), 13-15 Décembre 2005, Bangkok, Thailande.
   (b) *Distributed Node Location in clustered multi-hop wireless networks.* Nathalie Mitton, Éric Fleury. LOCALITY'05, 26 Septembre 2005, Cracovie, Pologne.
3. **Rapports de recherche :**
   (a) *Distributed Node Location in clustered multi-hop wireless networks.* Nathalie Mitton et Éric Fleury. RR-5723. Octobre 2005.
4. **Séminaires, présentations, exposés :**
   (a) *Localisation dans les réseaux sans fil multi-sauts grandes échelles.* Nathalie Mitton, Éric Fleury. Séminaire ACI Pair à Pair - Arcachon - France - 5-6 Septembre 2005.

Chapitre 6

Conclusion et perspectives

6.1 Conclusion

Mon objectif, au travers de cette thèse, a été de développer une solution d'utilisation des réseaux sans fil sur de larges échelles afin de répondre aux besoins naissants de notre société. Pour cela, il me paraissait important d'étudier les différentes contraintes de tels réseaux avant de pouvoir proposer une solution viable et fonctionnelle, même sur de grandes échelles.

J'ai pour cela proposé une solution distribuée qui se décompose en plusieurs étapes. Chacune des étapes répond à une application de tels réseaux tout en en considérant les contraintes. Chaque algorithme est né d'une étude des contraintes ou des structures et se compose d'algorithmes distribués, auto-stabilisants et robustes, nécessitant peu de ressources en énergie, bande passante ou taille mémoire. Toutes ces étapes ont été étudiées analytiquement et par simulation. Chaque solution a été comparée à d'autres solutions proposées dans la littérature et s'avère soit offrir de meilleurs résultats, soit de meilleurs compromis.

La première étape permet de structurer logiquement le réseau en clusters. A partir de l'étude de la structure ainsi formée, plusieurs caractéristiques ont pu être dégagées. Ces caractéristiques ont alors guidé la conception des algorithmes des étapes suivantes. La seconde étape a créé des liens entre ces clusters afin de pouvoir effectuer des diffusions générales d'information de façon efficace. L'algorithme de diffusion en soi est simple, il sélectionne un sous-ensemble de nœuds autorisés à relayer le message. Cette simplicité intrinsèque est d'autant plus riche qu'elle utilise une structure existante, sans en créer de nouvelles. L'algorithme de diffusion économise ainsi des messages et des ressources.

Enfin, la dernière étape permet à deux entités individuelles de communiquer, quelles que soient leurs positions dans le réseau. Ce processus de routage présente plusieurs originalités. En effet, dans un premier temps, il considère une approche inverse à

celle proposée jusqu'à maintenant afin de préserver une faible taille mémoire sur les nœuds. Ensuite, il emprunte des solutions algorithmiques à d'autres domaines comme les tables de hachage distribuées issues du domaine du Pair à Pair et le routage par intervalle issu des réseaux filaires.

6.2 Perspectives

Les différents algorithmes proposés dans chaque étape de ma proposition présentent tous des caractéristiques qui peuvent être étudiées plus en profondeur ou des points qui mériteraient certaines optimisations. Par exemple, le comportement de chaque algorithme pourrait être analysé dans un environnement plus mobile. Un autre point important est que toutes les analyses réalisées sur les différents algorithmes de ma proposition ont été conduites en supposant une couche MAC idéale et un modèle "Unit Disk Graph" pour définir le voisinage des nœuds. Considérer une couche MAC idéale permet de n'étudier que les caractéristiques propres à l'algorithme considéré, ce qui était le but recherché dans cette thèse. Cependant, un protocole de niveau 3 est utilisé conjointement avec des protocoles de niveaux inférieurs. De même, l'aire de transmission d'un nœud est rarement un cercle puisque la propagation des ondes dépend du milieu et des obstacles que peut rencontrer le signal. Il serait donc intéressant d'évaluer les différents algorithmes proposés dans ce manuscrit, en considérant différents protocoles pour les couches inférieures et différents modes de propagation des ondes suivant les environnements, aussi bien en ce qui concerne les analyses par simulation que les analyses théoriques. En effet, dans nos analyses théoriques, nous avons également considéré le modèle "Unit Disk Graph" mais d'autres modèles d'étude ont été proposés comme par exemple dans [27] ou [9] qui considère le modèle CDMA utilisé dans 802.11. De plus, comme dans un environnement sans fil réel, de nombreux paramètres d'environnement entrent en jeu pour assurer ou non l'existence d'un lien, on peut également considérer un modèle où un lien existe avec une certaine probabilité.

Par ailleurs, les réseaux sans fil couvrent une large famille de réseaux, comme les réseaux *ad hoc* ou les réseaux de capteurs. Bien que possédant tous des caractéristiques semblables, chacun a une utilisation plus précise et des propriétés supplémentaires. La solution d'utilisation grande échelle que j'ai décrite ici est générique et peut s'appliquer à tout type de réseau sans fil. Cependant, il me semble que pour concevoir un réseau efficace, il faut également considérer l'application du réseau. Ainsi, j'aimerais par la suite considérer plusieurs applications plus ciblées et un type de réseau plus précis comme par exemple les réseaux de capteurs.

En effet, les réseaux de capteurs offrent un certain nombre de défis et de verrous scientifiques. Ils sont le reflet de l'évolution à la fois des systèmes, des réseaux, de leurs composants mais aussi de leur organisation et des inter-actions et communications entre systèmes. Malgré des tailles souvent petites, ils intègrent une forte complexité, notamment de par l'infrastructure logicielle qui se retrouve distribuée à une grande échelle et qui se doit d'offrir des services d'auto-adaptabilité.

Les réseaux de capteurs sont de plus en plus déployés et offrent des perspectives nouvelles chaque jour. Aujourd'hui, on cherche à déployer un réseau de capteurs pour la surveillance des feux de forêts par exemple, afin de prévenir d'un incendie et d'intervenir avant qu'il n'ait causé trop de dégâts. À moyen terme, on cherche à intégrer ces capteurs dans les structures des bâtiments et ne les faire s'activer que si le bâtiment s'écroule afin de guider les secours. Les contraintes de base restent les mêmes : énergie et bande passante limitées. Cependant, avec les nouvelles applications et le déploiement de réseaux de plus en plus grands, d'autres points sont à considérer, comme la mobilité des capteurs (un réseau de capteurs peut être embarqué dans une voiture par exemple ou servir à traquer des animaux), la robustesse de la structure globale (il faut s'assurer que le réseau de base soit stable avant de penser à l'étendre) ou encore la sécurité. Pour une meilleure optimisation, tous ces aspects se doivent d'être étudiés conjointement.

La solution que j'ai développée dans cette thèse reste applicable, même si les capteurs présentent des propriétés supplémentaires par rapport au modèle générique des réseaux sans fil. Par exemple, lorsque l'on considère un réseau de capteurs, bien que la topologie reste dynamique du fait des apparitions ou disparitions de capteurs dues à la mort, à l'endormissement ou au réveil des entités, les capteurs ne se déplacent pas toujours. De même, les modèles de communication sont différents dans un réseau de capteur. Deux capteurs n'ont généralement pas besoin de communiquer directement. On observe surtout des communications de la station de base vers l'ensemble des entités (pour une synchronisation par exemple) ou des entités vers la station de base (pour retourner une mesure prise par un capteur par exemple). Les algorithmes que j'ai proposés peuvent être optimisés en fonction de ces caractéristiques plus précises.

Les contraintes à étudier, quelles qu'elles soient, doivent être considérées dans toutes les couches de communication, qu'il s'agisse de la conception au niveau signal, liaison de données ou routage. Tous les niveaux doivent considérer que les capteurs ont des capacités limitées en énergie et en taille mémoire, et non pas seulement la couche réseau. Tous ces défis rencontrés dans de tels réseaux ont été traités à différents niveaux. Les différentes propositions de protocoles de niveau physique cherchent à minimiser l'énergie dépensée en émission et en réception. Les protocoles de niveau MAC étudient des règles d'endormissement des nœuds afin de ne faire travailler qu'un sous-ensemble de capteurs à la fois. Les protocoles de routage cherchent à minimiser les inondations de découverte et maintenance des routes et à limiter le nombre de nœuds participant à ces diffusions. Cependant, toutes ces recherches d'optimisation restent indépendantes les unes des autres et ne sont pas toujours compatibles. Par exemple, les protocoles de routage peuvent avoir désigné certains capteurs pour diffuser un message que la couche MAC aura endormis. Dans ce cas, les messages ne seront pas relayés, ce qui peut avoir de graves conséquences suivant l'usage du réseau de capteurs. De la même façon, la couche la plus haute et la couche la plus basse ne peuvent être totalement décorrélées, les capteurs physiques devant répondre aux contraintes des applications.

Ainsi, chaque couche a besoin des informations des autres couches. Pour optimiser au maximum les communications dans les réseaux de capteurs, toutes ces couches ne peuvent rester indépendantes les unes des autres et travailler seules. Il faudrait parvenir à supprimer ce découpage en couche et à fusionner les qualités logicielles et matérielles des capteurs. Le routage et les applications doivent pouvoir être déterminés en fonction

des capacités physiques des capteurs et vice-versa. Les applications du réseau de capteurs doivent également guider les fonctionnalités qui doivent apparaître aux niveaux inférieurs. Ceci sera d'autant plus important quand on en viendra à faire communiquer des objets hétérogènes qui auront des fonctions de surveillance différentes dans le réseau. En effet, il faudra maintenir l'inter-opérabilité entre les composants et le bon fonctionnement global du réseau. Si toutes les couches sont fusionnées, les différences entre ces objets seront masquées à la vue des capteurs.

Bibliographie

[1] I. Abraham, D. Malkhi, and O. Dobzinski. LAND : Locality Aware Networks for Distributed Hash Tables. In *ACM-SIAM Symposium on Discrete Algorithms (SODA'04)*, New Orleans, LA, USA, 2004.

[2] K. Alzoubi, P. Wan, and O. Frieder. New distributed algorithm for connected dominating set in mobile *ad hoc* networks. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Hawaii, USA, January 2002.

[3] A. Amis and Prakash. Load-balancing clusters in wireless *ad hoc* networks. In *ASSET*, Richardson, Texas, USA, March 2000.

[4] A. Amis, R. Prakash, T. Vuong, and D. Huynh. Max-Min d-cluster formation in wireless *ad hoc* networks. In *INFOCOM*, Tel-Aviv, Israël, March 2000. IEEE.

[5] B. An and S. Papavassiliou. A mobility-based clustering approach to support mobility management and multicast routing in mobile *Ad-hoc* wireless networks. *International Journal of Network Management*, 11(6) :387–395, November 2001.

[6] F. Araujo, L. Rodrigues, J. Kaiser, L. Changling, and C. Mitidieri. CHR : A Distributed Hash Table for Wireless *Ad Hoc* Networks. In *4th International Workshop on Distributed Event-based Systems (DEBS'05)*, Columbus, Ohio, USA, June 2005.

[7] E. Baccelli. OLSR trees : A simple clustering mechanism for OLSR. In *MED-HOC-NET 05*, Porquerolles, France, June 2005.

[8] E. Baccelli and P. Jacquet. Flooding techniques in mobile *Ad-Hoc* networks. Technical Report RR-5002, INRIA, 2003.

[9] F. Baccelli, B. Błaszczyszyn, and M. Karray. Up and downlink admission/congestion control and aximal load in large homogeneous CDMA networks. In *Proc. of WiOpt*, Sophia Antipolis, France, 2003. to appear in MONET Special Issue on Optimization of Wireless and Mobile Networks 10(2), April 2005.

[10] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Communications of the ACM*, 46(2) :43–48, February 2003.

[11] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multihop wireless networks. In *INFOCOM*, Anchorage, Alaska, USA, April 2001.

[12] S. Basagni. Distributed clustering for *ad hoc* networks. In *International Symposium on parallel architectures algorithms and networks (I-SPAN'99)*, pages 310–315, Fremantle, Australia, June 1999.

[13] P. Basu, N. Khan, and T. Little. A mobility based metric for clustering in mobile *ad hoc* networks. In *Distributed Computing Systems Workshop (DISC)*, 2001.

[14] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1987.

[15] L. Blazevic, L. Buttyan, S. Capkun, S. Giordano, and J.-Y. Le Boudec. Self-organization in mobile ad-hoc networks : the approach of Terminodes. *IEEE Communications Magazine*, 39(6) :166–174, June 2001.

[16] L. Blazevic, S. Giordano, and J.-Y. Le Boudec. Self-organized Terminode routing. *Journal of Cluster Computing*, 5(2), April 2002.

[17] S. Capkun, M. Hamdi, and J.-P. Hubaux. GPS-free positioning in mobile ad-hoc networks. In *Proceedings of The 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, USA, January 2001.

[18] J. Cartigny. *Contributions à la diffusion dans les réseaux* ad hoc. PhD thesis, LiFL, Lille, December 2003.

[19] J. Cartigny, F. Ingelrest, and D. Simplot-Ryl. RNG relay subset flooding protocol in mobile ad-hoc networks. *IJFCS*, pages 253–265, 2003.

[20] M. Chatterjee, S. K. Das, and D. Turgut. WCA : A weight based distributed clustering algorithm for mobile *ad hoc* networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5(2) :193–204, April 2002.

[21] G. Chelius, E. Fleury, B. Sericola, and L. Toutain. On the NAP Protocol. Technical Report 5701, INRIA, 2005.

[22] B. Chen and R. Morris. L+ : Scalable landmark routing and address lookup for multi-hop wireless networks. MIT LCS technical report 837, MIT, March 2002.

[23] G. Chen, F. Garcia, J. Solano, and I. Stojmenovic. Connectivity-based k-hop clustering in wireless networks. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, Hawaii, USA, January 2002.

[24] C. Chiang, H. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *ICCS/ISPACS'96*, Singapore, November 1996.

[25] T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum, and L. Viennot. Optimized Link State Routing Protocol, October 2003. RFC 3626.

[26] F. Dai and J. Wu. Distributed dominant pruning in *ad hoc* networks. In *ICC'03*, Anchorage, Alaska, USA, May 2003.

[27] O. Dousse, F. Baccelli, and P. Thiran. Impact of interferences on the connectivity of *Ad Hoc* networks. In *Proc. of IEEE INFOCOM*, San Francisco, USA, 2003. to appear in IEEE Transactions on Networking.

[28] A. Ephremides, J. Wieselthier, and D. Baker. A design concept for reliable mobile radio networks with frequency hoping signaling. In *IEEE 75*, pages 56–73, 1987.

[29] P. Erdös and A. Renyi. *On Random Graphs*. Publicationes Mathematicae, 1959.

[30] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Scalable *ad hoc* routing : The case for dynamic addressing. In *INFOCOM*, Hong Kong, China, March 2004.

[31] L. M. Feeney and M. Nilsson. Investigating the energy consumption of a wireless network interface in an *ad hoc* networking environment. In *INFOCOM*, Anchorage, Alaska, USA, April 2001.

[32] Y. Fernandess and D. Malkhi. k-clustering in wireless *ad hoc* networks. In *ACM international workshop on Principles of mobile computing*, Toulouse, France, 2002.

[33] P. Fraigniaud and P. Gauron. An overview of the content-addressable network D2B. In *Proceedings of the 22nd ACM Symposium on Principles of Distributed Computing (PODC'03)*. ACM, July 2003.

[34] M. R. Garey and D. S. Johnson. *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman & Company, New York, 1979.

[35] M. Gerla and G. Pei. Fisheye State Routing Protocol (FSR). DRAFT draft-ietf-manet-fsr-02.txt, IETF, December 2001.

[36] M. Gerla and J. T.-C. Tsai. Multicluster, mobile, multimedia radio network. *ACM/Baltzer Journal of Wireless Networks*, 1(3) :255–265, July 1995.

[37] P. Gupta and P. Kumar. The capacity of wireless network. *IEEE Trans. On Information Theory*, 46(2) :388–404, March 2000.

[38] T. Herman and S. Tixeuil. A distributed TDMA slot assignment for wireless sensor networks. In *Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004)*, number 3121 in Lecture Notes in Computer Science, Turku, Finland, July 2004. Springer-Verlag.

[39] T.-C. Hou and T.-J. Tsai. An access-based clustering protocol for multihop wireless *ad hoc* networks. *IEEE Journal on Selected Areas in Communications*, 19(7) :1201–1210, July 2001.

[40] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot. Performance analysis of OLSR multipoint relay flooding in two *ad hoc* wireless network models. Technical Report RR-4260, INRIA, September 2001.

[41] P. Jacquet, A. Laouiti, P. Minet, and L. Viennot. Performance of multipoint relaying in *ad hoc* mobile routing protocols. In *Networking*, Pisa, Italy, 2002.

[42] M. Jiang, J. Li, and Y. Tay. Cluster Based Routing Protocol (CBRP). DRAFT draft-ietf-manet-cbrp-spec-01.txt, IETF, July 1999.

[43] D. Johnson, D. A. Maltz, and Y.-C. Hu. Dynamic Source Routing (DSR), February 2003.

[44] L. Kai and L. Jiandong. Mobile cluster protocol in wireless *ad hoc* networks. In *International Conference on Communication Technology (ICCT'2000) Proceedings*, August 2000.

[45] P. Krishna, N. H. Vaidya, M. Chatterjee, and D. K. Pradhan. A cluster based approach for routing in dynamic networks. In *ACM SIGCOMM*, pages 49–65. ACM, April 1997.

[46] B.-J. Kwak, N.-O. Song, and L. Miller. On the scalability of *ad hoc* networks. *Communications Letters, IEEE*, 8 :503– 505, 2004.

[47] T. J. K. Kwon and M. Gerla. Efficient flooding with passive clustering (PC) - an overhead-free selective forward mechanism for *ad hoc*/sensor networks. *Proceedings of IEEE*, 91 :1210–1220, 2003.

[48] J. Li, R. Morris, J. Jannotti, D. S. Decouto, and D. R. Karger. A scalable location service for geographic *ad hoc* routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (Mobicom'00)*, pages 120 – 130. ACM, August 2000.

[49] H. Lim and C. Kim. Multicast tree construction and flooding in wireless *ad hoc* networks. In *ACM MSWiM Workshop at MobiCom 2000*, Boston, MA, USA, August 2000.

[50] C. R. Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7) :1265–1275, 1997.

[51] H.-C. Lin and Y.-H. Chu. A clustering technique for large multihop mobile wireless networks. In *Vehicular Technology Conference (VTC'00)*, Tokyo, Japan, May 2000.

[52] D. Malkhi, M. Naor, and D. Ratajczak. Viceroy : A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2002.

[53] B. Mans and N. Shrestha. Performance evaluation of approximation algorithms for multipoint relay selection. In *The Third Annual Mediterranean Ad Hoc Networking Workshop, MED-HOC-NET 04*, Bodrum, Turkey, June 2004.

[54] P. Maymounkov and D. Mazières. Kademlia : A peer-to-peer information system based on the XOR metric. In *Electronic Proceedings for the 1st International Workshop on Peer-to-Peer Systems (IPTPS '02)*, MIT Faculty Club, Cambridge, MA, USA, March 2002.

[55] N. Mitton, E. Fleury, I. Guérin-Lassous, B. Séricola, and S. Tixeuil. On fast randomized colorings in sensor networks. Research Report LRI-1416, LRI, June 2005.

[56] R. Morris, J. Jannotti, L. Jinyang, and D. S. J. Decouto. Carnet : A scalable *ad hoc* wireless network system. In *Proceedings of the 9th ACM SIGOPS European Workshop : Beyond the PC : New challenges for the operating system*, September 2000.

[57] E. T. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *INFOCOM*, New-York, USA, June 2002.

[58] D. Niculescu and B. Nath. Ad hoc positioning system (APS). In *Proceedings of GLOBECOM'01*, November 2001.

[59] N. Nikaein, H. Labiod, and C. Bonnet. DDR-distributed dynamic routing algorithm for mobile *ad hoc* networks. In *MobiHoc*, Boston, MA, USA, November, 20th 2000.

[60] T. Ohta, S. Inoue, and Y. Kakuda. An adaptive multihop clustering scheme for highly mobile *ad hoc* networks. In *IEEE International Symposium on Autonomous Decentralized Systems (ISADS'03)*, pages 293–300, April 2003.

[61] C. Perkins. *Ad hoc networking.* Addison-Wesley, 2001.

[62] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-demand Distance Vector Routing, July 2003. RFC 3561.

[63] J. Polastre, R. Szewczyk, and D. Culler. Telos : Enabling ultra-low power wireless research. In *IPSN/SPOTS'05*, Los Angeles, CA, USA, April 2005.

[64] H. Pucha, S. M. Das, and Y. C. Hu. Ekta : An efficient DHT substrate for distributed applications in mobile ad hoc networks. In *WMCSA*, pages 163–173, 2004.

[65] A. Qayyum, L. Viennot, and A. Laouiti. Multipoint relaying : An efficient technique for flooding in mobile wireless networks. In *HICSS'02*, Hawaii, USA, January 2002.

[66] R. Rajaraman. Topology control and routing in *ad hoc* networks : a survey. *ACM SIGACT News*, 33(2) :60–73, June 2002.

[67] A. Ramalingam, S. Subramani, and K. Perumalsamy. Associativity-based cluster formation and cluster management in *ad hoc* networks. In *9th International conference on high performance computing (HiPC'02)*, Bangalore, India, December 2002.

[68] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[69] R. Riedi, P. Druschel, Y. C. Hu, D. B. Johnson, and R. Baraniuk. SAFARI : A self-organizing hierarchical architecture for scalable *ad hoc* networking networking. Research report TR04-433, Rice University, February 2005.

[70] A. Rowstron and P. Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Heidelberg, Germany, November 2001.

[71] C. Santivanez, B. McDonald, I. Stavrakakis, and R. R. Ramanathan. On the scalability of *ad hoc* routing protocols. In *INFOCOM*, New-York, USA, June 2002.

[72] N. Santoro and R. Khatib. Labeling and implicit routing in networks. *The computer Journal*, 28 :5–8, 1985.

[73] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord : A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (Sigcomm'01)*, pages 149–160. ACM Press, 2001.

[74] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE TPDS*, 13(1), January 2002.

[75] I. Stojmenovic and J. Wu. Broadcasting and activity scheduling in *ad hoc* networks. *IEEE Mobile Ad Hoc Networking*, pages 205–229, 2004.

[76] D. Stoyan, S. Kendall, and J. Mecke. *Stochastic geometry and its applications, second edition*. John Wiley & Sons, 1995.

[77] J. Van Leeuven and R. Tan. Interval routing. *The computer Journal*, 30 :298–307, 1987.

[78] A. C. Viana, M. Dias de Armorim, S. Fdida, and J. Ferreira de Rezende. Indirect routing using distributed location information. In *PERCOM '03 : Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, page 224, Washington, DC, USA, 2003. IEEE Computer Society.

[79] A. C. Viana, M. Dias de Armorim, S. Fdida, and J. Ferreira de Rezende. Self-organization in spontaneous networks : the approach of DHT-based routing protocols. *Ad Hoc Networks Journal*, 2005.

[80] J. Wu and H. Li. A dominating set based routing scheme in *ad hoc* wireless networks. *Telecommunication Systems*, pages 13–36, 2001.

[81] J. Wu and W. Lou. Forward node set based broadcast in clustered mobile *ad hoc* networks. *Wireless Communications and Mobile Computing*, 3(2) :141–154, 2003.

[82] J. Y. Yu and P. H. Chong. 3hBAC (3-hop between adjacent clusterheads) : A novel non-overlapping clustering algorithm for mobile *ad hoc* networks. In *PacRim'03*, Victoria, Canada, August 2003.

[83] H. Zhai, Y. Kwon, and Y. Fang. Performance analysis of IEEE 802.11 MAC Protocols in wireless LANs. *Wireless communications and mobile computing*, 4 :917–931, 2004.

[84] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry : A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1), January 2004.