Topologie Virtuelle pour Réseaux Hybrides

Fabrice Theoleyre, Fabrice Valois

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Topologie Virtuelle pour Réseaux Hybrides

Fabrice Theoleyre — Fabrice Valois

N° 5035

Décembre 2003

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5035--FR

Topologie Virtuelle pour Réseaux Hybrides

Fabrice Theoleyre* , Fabrice Valois†

Thème 1 — Réseaux et systèmes
Projet ARÈS

Rapport de recherche n° 5035 — Décembre 2003 — 68 pages

Résumé : La gestion de la mobilité dans les réseaux hybrides, réseaux sans fil multisauts connectés à Internet, reste un problème complexe. Il est en effet nécessaire que tout nœud d'un réseau ad hoc puisse joindre et être joignable d'Internet. La plupart des propositions existantes essaient de calquer les approches des réseaux filaires, bien qu'elles soient peu optimisées pour les réseaux sans fil multisauts. Dans ce mémoire, nous proposons en premier lieu la création d'un backbone virtuel. Il nous permet de réduire les dépenses en énergie, de hiérarchiser les rôles et d'optimiser les communications de broadcast problématiques dans les réseaux ad hoc. Nous intégrons de plus les points d'accès wifi à ce backbone afin de créer un prolongement naturel de la connexion Internet. Puis, dans un deuxième temps, nous regroupons les nœuds proches géographiquement pour créer de véritables zones de services, intégrées au backbone. Ces zones de services, avec des chefs attitrés, peuvent prendre en charge des fonctionnalités de localisation des mobiles, de paging, de diffusion des informations de topologie... Ces structures maintenues dynamiquement constituent un véritable socle pour développer de nouvelles fonctionnalités à destination des réseaux hybrides.

Mots-clés : réseaux adhoc, cluster, backbone (mcds), topologie virtuelle, maintenance

* fabrice.theoleyre@insa-lyon.fr
† fabrice.valois@insa-lyon.fr

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Virtual Toplogy for Hybrid Networks

Abstract: Hybrid networks are heterogeneous networks merging wireless nodes, ad hoc nodes and where the interconnection to IP world is an important topic through gateways called AP (access point). Indeed, each node can be contacted and can contact another node in Internet. To reach that, architectures to support mobility management will be study. The solutions inspired by wired networks are not particularly suited to hybrid networks. We propose to use a virtual dynamic infrastructure including both backbone and clusters. A backbone is suited to spare energy, optimize control traffic diffusion, hierarchize participants,... The clusters are intended to create services areas, and to handle particularly the mobility management. We present algorithms to both construct and maintain these structures. This dynamic topology is robust according to mobility, and is well suited to implement mobility management and localization procedure. Finally, the number of backbone members and clusters are completely parameterizable according to the environment.

Key-words: ad hoc networks, cluster, backbone (mcds), virtual topology, maintenance

Contents

Introduction

Les réseaux ad hoc sont parfois définis comme des réseaux spontanés sans fil [41]. Ils réunissent un grand nombre d'objets communicants sans fil, sans infrastructure et tous ces objets peuvent se déplacer. De tels réseaux sont donc intrinséquement différents des réseaux classiques qui utilisent une dorsale filaire et des collecteurs de trafic pour connecter plusieurs réseaux locaux filaires ou sans fil. Les réseaux ad hoc doivent s'auto-organiser pour acheminer le trafic d'un point à l'autre du réseau ad hoc. L'auto organisation passe d'abord par une solution d'acheminement du trafic, puisque la source et la destination peuvent ne pas être à portée radio. Le réseau doit donc collaborer avec de potentiels nœuds intermédiaires, s'auto attribuer des adresses... Toutes les fonctionnalités doivent à terme se déployer automatiquement sans paramétrage éventuel de l'utilisateur.

Les réseaux ad hoc présentent un intérêt important dans les environnements sans infrastructures. Par exemple, ils peuvent être utilisés par les militaires sur les champs de batailles pour établir des liaisons entre les différentes unités mobiles : les communications avec la base étant souvent prises en charge par des liaisons satellitaires. Les réseaux ad hoc peuvent également être utiles à l'organisation des secours après une catastrophe naturelle où toutes les infrastructures de communication ont été détruites, pour rechercher des survivants après une avalanche... A terme, ils pourront être déployés pour des applications plus classiques comme les conférences ou pour l'établissement de petits réseaux sans fil bon marché mais également pour des applications plus spécifiques comme l'extension spontanée de couverture radio par collaboration de nœuds ad hoc...

Les réseaux ad hoc présentent des spécificités fortes. Tout d'abord, les clients de tels réseaux peuvent être constitués d'ordinateurs de bureau, de PC portables, d'assistans personnels (PDA), mais également à terme de téléphones portables ou tout objet communicant. De tels types de terminaux engendrent immédiatement des contraintes en terme d'énergie, d'hétérogénéité, de puissance de calcul, de performances matérielles. De plus, un réseau ad hoc ne possédant pas d'infrastructures, qu'elle soient fixes ou mobiles, les fonctionnalités réseaux (e.g. routage, localisation, etc.) doivent être prise en charge par les terminaux, introduisant la notion de nœud-routeur pour tout terminal. Chaque nœud doit être capable de maintenir une vue partielle du réseau qui lui permettra de transmettre ses informations, en sachant que le destinataire potentiel peut se trouver à plusieurs sauts, et donc qu'il doit obligatoirement passer par d'autres nœuds tels que lui. Tout nœud doit donc participer de façon collaborative à l'établissement et à la maintenance du réseau. A cet environnement multi-sauts à nœuds hétérogènes s'ajoute la mobilité potentielle de chaque nœud: tout terminal peut se déplacer dans une direction quelconque, obligeant une adaptation dynamique des capacités du réseau.

Le routage dans les réseaux ad hoc constitue un thème essentiel de recherche [24]. Il faut avant tout trouver un moyen pour router les données dans le réseau de façon efficace. Il faut donc économiser la bande passante, ressource rare en radio, limiter le nombre de

collisions. Il est également indispensable de concevoir un protocole efficace lorsque le nombre de participants et leur mobilité respective augmentent. Un protocole répond généralement à un certain nombre de contraintes, rarement à toutes de façon efficace. Il faut donc quantifier le comportement de chaque protocole face à ces différents critères, ce à quoi s'attachent les auteurs de [46].

Il existe deux grands types de protocole de routage. Les proactifs maintiennent des informations globales sur le réseau avant toute utilisation. L'un des plus anciens, WRP[1] [38], se borne à échanger des tables de routage. Cependant, il existe des solutions récentes plus performantes, par exemple OLSR[2] [20], un des protocoles proactifs en cours de standardisation par l'IETF[3] au sein du groupe de travail MANET[4] : un nœud choisit dans son voisinage un groupe de nœuds (les *Multi Points Relais*) chargé de relayer ses paquets de contrôle, afin que tous les autres nœuds soient atteints. Un nœud ne déclare que les liens l'unissant à ses MPR. Les performances sont donc améliorées puisque seuls certains nœuds retransmettent l'information, limitant la redondance d'informations, et économisant la bande passante.

Il existe ensuite les protocoles de routage réactifs qui ne demandent des informations que lorsqu'ils en ont besoin. DSR[5] [32] et AODV[6] [42], les deux protocoles réactifs en cours de standardisation à l'IETF, effectuent des requêtes de routes, inondées dans tout le réseau lorsqu'un nœud a besoin de transmettre des informations. Les nœuds intermédiaires enregistrent un état temporaire leur permettant de faire suivre la réponse, s'il y en a une, vers la source. Lorsque cette requête atteint la destination, cette dernière répond et crée une route au sein de chaque nœud intermédiaire. Il faut donc mettre en place un mécanisme complexe de cache et de maintenance de routes [1, 57] pour reconstruire localement les routes si elles sont cassées par un changement de topologie.

Cependant, certaines approches, appelées hybrides, tentent de combiner ces deux approches. ZRP[7] [28, 30] par exemple maintient des informations proactives au sein d'une zone locale. Les communications locales sont donc optimisées. Ensuite, un nœud va envoyer des requêtes de route lorsqu'il ne connaît pas la destination. Au lieu d'inonder le réseau, il va envoyer la requête aux nœuds en périphérie de zone, les *nœuds bordure*. Ceux ci vont ensuite relayer à leurs propres nœuds bordure, jusqu'à que la destination soit trouvée, la réponse de route empruntant la suite de nœuds bordure utilisée à l'aller. Cependant, il faut prendre quelques précautions lors du relais de requêtes pour ne pas écrouler les performances en relayant plusieurs fois la même requête [29]. Il existe notamment une proposition d'auto-adaptation de certains paramètres aux caractéristiques de l'environnement [40], notion rarement prise en compte dans les propositions de protocoles de routage. Certains autres axes de recherche ont besoin d'être développés. La sécurité est par exemple néces-

[1] Wireless Routing Protocol
[2] Optimized Link State Routing Protocol
[3] Internet Engineering Task Force, organisme de standardisation des protocoles d'Internet
[4] Mobile Ad hoc NETworks
[5] Dynamic Source Routing
[6] Ad Hoc On Demand Distance Vector
[7] Zone Routing Protocol

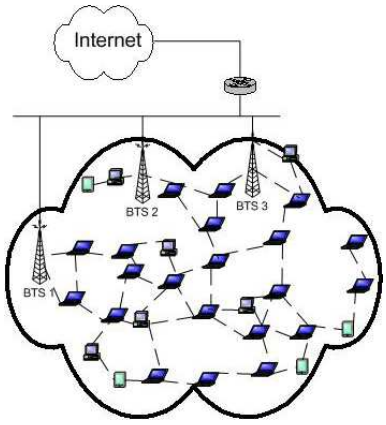

Figure 1: Exemple de réseau hybride

saire à la démocratisation de tels réseaux. La qualité de service constituerait également un plus dans des réseaux à relativement faible débit.

A côté des problématiques du routage, vient se poser la question de l'interconnexion d'un réseau ad hoc à un réseau local filaire ou à Internet. Un tel type de réseau est appelé communément réseau hybride (fig.1). Les entreprises pourraient ainsi installer un réseau d'accès sans fil pour leur personnel itinérant, les opérateurs augmenter leur couverture en réduisant leurs coûts, la domotique trouver un médium de communication adéquat... Cependant, une telle interconnexion n'est pas triviale : il faut gérer les communications entrant et sortant de la zone ad hoc. Les communications peuvent potentiellement passer par plusieurs points d'accès. Il nous faut donc une solution pour gérer la mobilité : un nœud se déplaçant au sein du réseau et changeant de point d'accès doit continuer à pouvoir communiquer avec l'extérieur et être joignable par un hôte présent sur Internet. De plus, les communications doivent également être optimisées au sein de la zone ad hoc vers les points d'accès. Il existe actuellement des solutions pour les réseaux sans fil à un saut possédant une dorsale filaire. Ces solutions proposent des réponses en terme d'adressage global, de changement de point d'accès, de gestion de la localisation des mobiles, de notification (paging) pour les mobiles endormis... Il serait donc intéressant de pouvoir s'inspirer de telles solutions pour les réseaux hybrides. Cependant, de tels réseaux, comme nous l'avons dit, ne comprennent pas d'infrastructures fixes.

Nous proposons donc de construire un **backbone virtuel** dynamique afin d'acheminer le trafic de contrôle de façon fiable et efficace. Puis, sur cette dorsale, nous allons ajouter un regroupement géographique de nœuds en **clusters** afin de gérer leur localisation, diffuser les paramètres d'accès au réseau hybride... Une telle structure devra s'adapter dynamiquement aux changements de l'environnement et permettra donc de déployer sur ces réseaux hybrides certaines fonctionnalités des réseaux filaires, telles que la localisation.

La section 1 dressera un état de l'art pour les solutions de construction d'arbres et de clusters dans un réseau. La section 2 motivera l'intérêt de l'établissement d'une structure d'arbre et de clusters puis détaillera les détails de la construction et de la maintenance de la topologie virtuelle proposée. La section 3 exposera les résultats des simulations implémentées. Enfin, la section 4 concluera ce travail et exposera plusieurs perspectives.

1 État de l'art

1.1 Modélisation

Nous pouvons modéliser un réseau hybride sous la forme d'un graphe orienté. Les sommets seront représentés par les terminaux du réseau. Si un nœud A possède un lien radio vers le nœud B, il existera dans le graphe de modélisation un arc orienté du sommet représentant A vers le sommet représentant B. Un arc correspond donc à un saut radio. Un voisin de A est un nœud possédant un lien radio avec A.

Nos avons introduit par soucis de clarté les notations suivantes :

- V : ensemble des nœuds du graphe
- n : cardinalité du réseau/graphe ($= |V|$)
- E : ensemble des liens/arcs du graphe
- n : nombre de liens ($= |E|$)
- $\text{chemin}_{u \to v}$: suite de sommets pour aller de u à v
- N_k(u) : ensemble des k-voisins de u, i.e. l'ensemble des nœuds à au plus k sauts de u
- Δ : degré d'un nœud, i.e. le nombre de voisins
- Δ_k : k-degré d'un nœud, i.e. le nombre de ses k-voisins
- AP : point d'accès d'un réseau hybride, passerelle entre le filaire et la zone ad hoc

Par simplification, nous supposons les liens radios bidirectionnels. Notre réseau hybride peut donc être représenté par un graphe non orienté composé d'arêtes. Les liens unidirectionnels ne sont donc pas pris en compte par notre solution, un processus étant chargé de les distinguer des liens bidirectionnels, et de les éliminer.

Les réseaux ad hoc sont souvent modélisés par un *unit disk graph* [37]. Dans un tel graphe, la portée radio d'un nœud quelconque est circulaire, de distance unitaire. Le nombre de voisins indépendants[8] dans un *unit disk graph* étant borné à 5, cette propriété permet à de nombreux auteurs de pouvoir borner la taille des structures de graphes qu'ils proposent.

[8] L'ensemble des voisins indépendants d'un nœud N est l'ensemble maximum ϵ de ses voisins, qui ne sont pas voisins avec un autre nœud de l'ensemble ϵ.

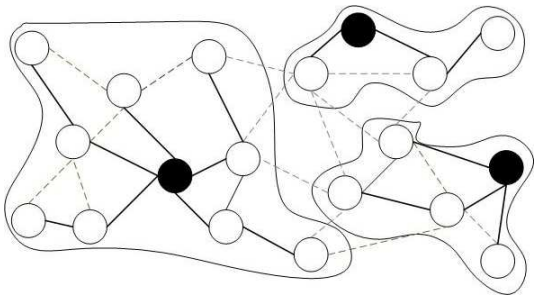

Figure 2: Exemple de 2-Clusters

Un tel graphe permet donc de modéliser une première approximation des réseaux ad hoc pour une étude analytique. Cependant, des nœuds pouvant avoir une portée différente, il est important de proposer une solution robuste à l'hétérogénéité de porteé radio dans le réseau.

1.2 Clusters

1.2.1 Intérêt

Un réseau hybride est par définition un réseau à plat : tout nœud est considéré comme égalitaire par rapport aux autres, exception faite du point d'accès. Cependant, il serait intéressant de reproduire l'architecture des réseaux d'accès sans fil classique, un point d'accès couvrant une zone de service. Le découpage en zones du réseau hybride, avec l'élection d'un chef de zone, permettrait donc de créer des zones de services, avec des *Points d'accès virtuels*, gérant l'adressage, la localisation, la distribution du trafic de contrôle ...

1.2.2 Définition

La clusterisation d'un réseau est son découpage en zones de diamètre constant. Formellement, chaque cluster comprend un *clusterhead*, sorte de centre du cluster. Ensuite, tout nœud du réseau doit être voisin d'un *clusterhead*. La clusterisation est donc l'élection des nœuds devenant clusterhead. Usuellement, les algorithmes de clusterisation cherchent à minimiser le nombre de clusterheads élus afin d'obtenir un nombre réduit de clusters.

Par extension, un *k-cluster* oblige tout nœud à être à moins de k sauts d'un clusterhead (fig.2) :

$$\forall u \in V, \exists c \in C \ / \ c \in N_k(u) \ \ (C = \bigcup Clusterheads) \tag{1}$$

1.2.3 1-Clusters

Construction Il existe de nombreuses propositions de méthodes de 1-clusterisation dans les réseaux ad hoc, aisément adaptable aux réseaux hybrides. En effet, les clusters peuvent être utilisés dans de nombreuses applications comme l'économie d'énergie, la réservation de ressources, l'attribution de fréquences...

Les auteurs de [33] proposent la création de clusters recouvrants. Chaque nouveau nœud exécute l'algorithme de construction et maintenance de clusters à partir de la liste actuelle des clusters, fournie par ses nouveaux voisins. Il n'existe pas de clusterhead. L'algorithme ajoute tous les clusters possibles incluant le nouveau nœud, puis trie tous les clusters par ordre de taille. Enfin, il supprime les clusters redondants, privilégiant les clusters de taille importante, tentant ainsi de minimiser le nombre de clusters dans le réseau. Un cluster redondant est un cluster inclus dans un autre de taille supérieure. Ce nœud fournit à l'ensemble de ses voisins la nouvelle liste, propagée ensuite dans le réseau. Lorsqu'un nœud disparaît, un de ses voisins est élu pour tenter de fusionner des clusters existants. Cependant, cet algorithme crée un nombre de clusters important, la simultanéité des évènements n'est pas prise en compte, l'overhead se multiplie en cas d'environnement mobile, la table des clusters peut présenter des incohérences si deux nœuds du réseau la modifient simultanément...

Il existe une méthode très répandue de création de clusters [8, 15, 16, 36, 45, 53]. La première étape consiste à connaître son voisinage. Ensuite, chaque nœud prend la décision, selon sa connaissance locale de la topologie, de devenir clusterhead ou non. La décision est diffusée au voisinage, forçant les voisins du nouveau clusterhead qui ne sont pas encore affiliés à un cluster à le choisir comme clusterhead. La décision d'élection peut par contre prendre des formes très diverses. Souvent, cette décision est basée sur l'identifiant du nœud [36, 45] : le nœud possédant un identifiant maximum dans son voisinage durant un intervalle de temps Δt s'élit clusterhead. L'identifiant étant immuable au cours du temps, une telle méthode permet de stabiliser les clusterheads. Cependant, un tel choix peut être loin de l'optimum si un nœud à fort identifiant est très mobile, si sa réserve d'énergie est faible... De plus, l'élection n'est pas égalitaire, les nœuds à identifiant élevé consommant en moyenne plus d'énergie que les autres, puisqu'ils restent plus longtemps clusterheads. La complexité en message est en $O(n)$, puisque chacun envoie un seul message de décision. La complexité en temps est en $O(n)$, dans le pire cas d'une élection en interblocage où un seul nœud est élu à chaque étape. Les complexités sont identiques pour tous les algorithmes suivants de clusterisation, très similaires.

[8] propose de prendre une décision basée sur la mobilité. Seule la mobilité relative est intéressante. En effet, un nœud avec une faible mobilité relative gardera un voisinage plus stable au cours du temps, condition idéale pour être clusterhead, favorisant la stabilité des clusters. Chaque nœud mesure donc au cours du temps le niveau de signal qui l'unit à chacun de ses voisins. Le quotient de la puissance actuelle sur la puissance mesurée à l'intervalle de temps précédent est représentatif de la mobilité. En effet, l'atténuation de puissance peut être modélisée par une fonction de la distance selon une loi du type $d^{-\alpha}$, α étant l'atténuation caractéristique du milieu. Les auteurs introduisent ensuite la notion de valeur de mobilité agrégée de X, variance du quotient pour chacun des voisins de X de la puissance du signal

au temps t et au temps $t + \Delta t$. Enfin, les n dernières valeurs de cette mobilité agrégée sont stockées et moyennées pour donner la valeur de mobilité relative. L'introduction de la mobilité pour l'élection des clusters est très intéressante, cependant, il nous semble que d'autres critères peuvent rentrer en compte, ce que les auteurs ne proposent pas. De plus, cette estimation de la mobilité à partir de la puissance ne prend pas en compte certains phénomènes physiques tels que les évanouissements, les atténuations hétérogènes... Enfin, les auteurs ne décrivent aucune méthode de maintenance.

Enfin, les auteurs de [15, 16] présentent une décision basée sur plusieurs qu'ils jugent important pour une clusterisation efficace :

- Degré : la différence entre le degré effectif et un degré idéal, un clusterhead devant prendre en charge un nombre moyen de clients, ni trop faible pour ne pas être inutile, ni trop élevé afin de ne pas être saturé ;
- Distances : la somme des distances vis à vis des voisins, obtenues par GPS, un clusterhead devant être proche de ses voisins pour les garder le plus longtemps possible ;
- Mobilité : la mobilité relative, obtenue là aussi par monitoring de la puissance du signal ;
- Temps : temps pendant lequel il a été clusterhead, pour ne pas qu'un clusterhead s'épuise et introduire de l'équilibrage de charge.

Une telle approche nous semble intéressante, plusieurs critères étant plus représentatifs de la stabilité. Cependant, dans les simulations, le degré compte pour 70% du poids total, ce qui annihile quasiment l'effet des autres paramètres, le temps et la mobilité ne comptant chacun que pour 5%. Leurs simulations reviennent donc quasiment à une élection classique basée sur le degré.

[34] propose la création de clusters en supprimant tout paquet de contrôle spécifique à la clusterisation. La décision est prise suivant un champ de 2 bits présents dans tous les paquets. Il permet de traduite 4 états : *initial*, *clusterhead*, *passerelle* et *ordinaire*. Seuls les nœuds à l'état *initial* peuvent devenir clusterhead, ce qui oblige une distance minimale de 2 sauts entre clusterheads. Le nœud devenant clusterhead est le premier qui arrive à se déclarer comme tel. Après une période d'inactivité, tous les nœuds reviennent à l'état *initial*. Un nœud entendant au moins deux clusterheads devient *passerelle*, cet état devenant *ordinaire* dés qu'il n'en entend plus qu'un. Les auteurs proposent d'utiliser cette clusterisation uniquement pour la diffusion de paquets de contrôle, ce qui explique que les clusters disparaissent lorsqu'il n'y a plus de trafic. Il n'existe donc pas vraiment de procédure de maintenance. Cependant, cette disparition peut être problématique pour de nombreuses autres applications. D'autre part, le nombre de clusters et leur future stabilité ne nous semblent pas être optimisés par un tel algorithme.

Maintenance La maintenance est indispensable pour une structure dynamique. Les clusters doivent constamment être mis à jour pour que les clusterheads restent au sein de leur cluster, qu'un nœud mobile se réaffilie à un cluster ou en crée au contraire un nouveau.

[55] présente un rapide panorama des techniques de clusterisation et des comportements lors de la maintenance des algorithmes. Les auteurs présentent tout d'abord une technique basée sur un poids générique et une sur l'identifiant, techniques quasiment similaires à celles présentées précédemment. L'algorithme basé sur le poids oblige à maintenir continuellement en tant que clusterhead le nœud à plus fort poids du cluster. Il provoque donc des reconfigurations en cascade. Par contre, l'autre algorithme basé sur l'identifiant permet à un clusterhead de garder son rôle tant qu'il peut servir d'autres nœuds, un nœud ne changeant de clusterhead que lorsque son ancien chef est hors de portée. Ce deuxième algorithme présente donc une stabilité plus importante. Cependant, les auteurs comparent deux méthodes différentes de maintenance de clusters. Ainsi, le deuxième algorithme utilisant au lieu de l'identifiant un poids représentatif de la stabilité d'un nœud permettrait peut être de meilleures performances. Cependant, les auteurs n'abordent pas une telle question. Enfin, les auteurs présentent également une technique basée sur l'estimation de la probabilité de connexion au clusterhead. La probabilité est calculée pour chaque lien, et peut donc être calculée pour un chemin, suite de liens, dans le réseau. L'algorithme va tendre à choisir un clusterhead stable en demandant ceux disponibles et en choisissant celui maximisant la probabilité de connexion. Or les auteurs ne proposent pas de méthode de calcul d'une telle probabilité. Une telle estimation semble non triviale, cette probabilité dépendant de la vitesse, de la localisation, de la connexité, des intermédiaires entre le nœud et son clusterhead...

Dans [36], les auteurs ont choisi de ne pas élire un clusterhead, tous les nœuds étant égaux. Un nœud connaît son 2-voisinage. Il sait donc si le nœud de son cluster à plus fort degré se trouve encore à portée. Si ce n'est pas le cas, alors il doit changer de cluster. Il peut rejoindre un ancien cluster si tous les membres de ce cluster sont à moins de 2 sauts, sinon, il crée un nouveau cluster. Cette méthode permet de ne pas "casser" un cluster quand son clusterhead part, puisque le nœud à plus fort degré est automatiquement désigné comme clusterhead. Cependant, une telle solution oblige le nœud à plus fort poids à être le centre du cluster, ce qui implique donc une mise à jour plus fréquente des clusters. De plus, les auteurs utilisant uniquement le degré, le poids change fréquemment, ce qui oblige à remodeler constamment les clusters alors que la structure précédente était peut être toujours efficace. Enfin, une telle maintenance oblige à connaître pour un k-cluster son 2k-voisinage, ce qui introduit une surcharge réseau importante.

Les auteurs de [15, 16] proposent une méthode de maintenance plus complexe. Un nœud va surveiller le lien qui l'unit à son clusterhead. Si un tel lien est sur le point de se rompre, il va effectuer par analogie avec le GSM, un handover vers un autre clusterhead, les 2 clusterheads dialoguant pour prendre en charge le déplacement du nœud. Cependant, les détails d'un tel dialogue ne sont pas développés par les auteurs. Si un nœud s'est trop déplacé et se trouve hors de portée de tout clusterhead, le mécanisme d'élection est de nouveau exécuté pour obtenir un nouvel ensemble de clusterheads. Les auteurs montrent bien l'apériodicité

de la maintenance lors des simulations. Cependant, la maintenance requiert un nombre d'élections fréquentes, il nous semble donc que l'overhead induit peut être relativement important. De plus, le mécanisme de réélection devant être estimé de façon probabiliste, il est difficile d'étudier de façon analytique la complexité en message et temps d'une telle maintenance. Une simulation est donc nécessaire pour étudier l'overhead. De plus, les clusterheads pour effectuer un handover doivent connaître les routes les unissant et devraient mettre en place un mécanisme semblable à celui de Mobile IP. La diffusion d'information de contrôle serait facilitée par la mise en place d'un backbone reliant tous les clusterheads, comme nous le proposons.

[31] ne propose qu'une méthode de maintenance, la construction étant un cas particulier. Au début, un nœud va envoyer une `request-to-join` pour tenter de trouver un clusterhead candidat. S'il reçoit une réponse, il peut renvoyer un `join`. Sinon, il va tenter d'envoyer un `hello` pour devenir clusterhead. Les auteurs proposent comme optimisation qu'un nœud non attaché à un clusterhead qui reçoit un `request-to-join` attende un temps aléatoire avant d'en envoyer un lui même. Il se peut qu'une réponse passe par lui, et donc que sa requête n'ait jamais besoin d'être émise. Un nœud qui quitte son cluster doit impérativement envoyer un `disconnect` pour que son clusterhead détruise les informations temporaires qu'il a en sa possession. Pour éviter que le nombre de clusterheads augmente au cours du temps, il existe un mécanisme de fusion de clusters : un clusterhead étant tout seul dans son cluster va tenter d'envoyer un `request-to-join` pour savoir s'il ne peut pas être pris en charge. Cependant cette méthode, selon nous, n'optimise pas le nombre de clusterheads, puisque plusieurs clusterheads peuvent par exemple se déclarer en même temps. De plus, seuls les clusters ne contenant qu'un seul clusterhead peuvent fusionner, ce qui a tendance à surestimer le nombre de clusters nécessaires.

1.2.4 Applications

[34] propose, comme nous l'avons vu, de créer un ensemble de clusters avec passerelles. Il existe ici une clusterisation dédiée à la diffusion du trafic de contrôle, les clusters disparaissant lorsqu'il n'existe plus de trafic à envoyer. Pour éviter la redondance des retransmissions et les collisions, les auteurs proposent que seules les passerelles et les clusters relaient les paquets de contrôle, les nœuds *ordinaires* ne faisant que les réceptionner. Les auteurs créent des clusters à la suite de l'envoi d'un paquet de contrôle. Avec ces clusters sont créées des passerelles. Les passerelles sont les nœuds voisins de nœuds appartenant à d'autres clusters, et qui n'entendent pas plus de n autres passerelles dans le voisinage, n dépendant également du nombre de clusterheads entendus. Seules les passerelles ont le droit de relayer à d'autres clusters les paquets de broadcast. Les auteurs utilisent donc le clustering pour la limitation du flooding, les passerelles trop redondantes étant supprimées. Il n'existe aucune maintenance, les clusters étant créés à la volée à chaque envoi du paquet de contrôle initiateur, qui peut être une *Route Request* d'AODV...

Les auteurs de [31] créent des clusters dans lesquels chaque nœud réserve des ressources pour envoyer ses flux. Chaque trame est découpée en m mini-slots de contention pour départager les nœuds souhaitant accéder au médium, et un slot de message. Durant un

mini-slot, chaque émetteur fait une demande d'accès. Si un nœud détecte plusieurs de ces demandes, il va envoyer un avertissement, les émetteurs tentant alors de nouveau leur demande avec une probabilité p dans le mini slot suivant. Les clusters doivent donc être de diamètre réduit afin de proposer des mini slots de contention les plus courts possibles (si deux émetteurs sont à maximum k sauts, il faut attendre O(k) temps pour être sûr que deux nœuds n'ont pas effectué une demande).

Dans [36], les auteurs assignent un code CDMA différent pour deux clusters voisins. Il n'existe donc pas d'interférences entre clusters. Puis, ils mettent en place un TDMA pour que chaque nœud puisse réserver des slots de temps pour ses communications. Chaque émetteur va donc réserver un slot et donner la périodicité de ses envois. Le premier paquet va donc réserver les ressources le long du chemin traversant les clusters. Chaque nœud acceptant la réservation va acquitter le paquet et se préparera à écouter sur le bon canal au temps indiqué pour recevoir le paquet suivant.

Les clusters peuvent naturellement servir au routage. Dans [33], les nœuds bordure informent leur cluster des nœuds adjacents. Un algorithme du plus court chemin, par exemple celui de Dijkstra, peut être exécuté sur ce réseau de nœuds bordure. Des messages de contrôle échangés entre les différents clusters permettent de connaître tous les nœuds bordure du réseau. La maintenance des routes se fait par la maintenance des informations sur les clusters et nœuds bordure. Un clusterhead calculant toutes les routes et stockant les informations sur le réseau pourrait sans doute réduire la charge des autres nœuds. Ceci serait d'autant plus intéressant que seuls les clusterheads s'échangeraient des paquets de topologie des clusters et donc, ceci réduirait l'overhead. [54] présente une étude analytique de l'overhead induit par les solutions de routage hiérarchiques. Les auteurs étudient de façon quantitative l'influence du nombre de nœuds par cluster, de niveaux de hiérarchie, de cassures de liens, du degré...

Les solutions utilisant les clusters sont extrêmement diverses puisqu'elles touchent au domaine du routage, de la réservation de ressources... Les possibilités d'un regroupement logique de nœuds sont nombreuses et constituent encore un vaste champ d'études.

1.2.5 k-Clusters

Alors qu'il existe de nombreux algorithmes de 1-clusterisation, ceux adaptés aux k-clusters sont relativement rares. Pourtant, les k-clusters permettent de développer certaines fonctionnalités plus aisément, comme nous le verrons dans la partie 2. Nous allons donc présenter les principales propositions traitant d'un tel sujet.

Les auteurs de [25] ont une approche originale de k-clusterisation. Leur algorithme se décompose en 2 étapes. En premier lieu, les auteurs créent, à partir de leur graphe, un arbre en utilisant un algorithme de *spanning tree*, le spanning tree pouvant être remplacé par un MCDS[9]. Cependant, il est préférable que deux nœuds proches géographiquement soient également proches dans l'arbre. Sinon les clusters finaux risqueront de réunir des nœuds

[9] *Minimum Connected Dominating Set* : classiquement utilisé pour modéliser les backbones dans les réseaux ad hocs. Cf. paragraphe 1.3

relativement distants. Les auteurs proposent d'utiliser l'algorithme de Marathe. Ce dernier crée tout d'abord un spanning tree quelconque. L'ensemble des nœuds est ensuite découpé en ensemble S_i selon la hauteur i des nœuds dans l'arbre. On procède de façon itérative en choisissant des dominants de S_i qui couvrent le plus de nœuds dans S_i. Les nœuds de S_{i-1}, parents de nouveaux dominants non connexes à l'arbre, peuvent être choisis pour interconnecter l'ensemble des dominants. Dans un deuxième temps, l'algorithme va étayer des branches pour regrouper les branches en ensembles de k nœuds. Chaque nœud va donner son poids à son père dans l'arbre. Le père va calculer son propre poids en additionnant le poids de chacun de ses fils tel que : $Poids = 1 + \sum_{i=1}^{nbfils} Poids_{fils_i}$. Un tel poids représente donc le nombre de nœuds présents sous un nœud de l'arbre. Un père peut donc étayer cette branche si ce poids dépasse un maximum, le fils direct devenant clusterhead. La complexité de la première étape dépend de l'algorithme de MCDS choisi. La deuxième étape nécessite, selon nos calculs, $O(2 \cdot n)$ messages puisque chaque nœud envoie au maximum un message à son père et reçoit au maximum un message de son père. La complexité en temps est en $O(n)$ puisqu'il faut que chaque message soit transmis au supérieur avant que celui ci prenne toute décision, et que la longueur de l'arbre est de n dans le cas pire d'un arbre à une seule feuille (chaîne linéaire de nœuds...). Cependant, les auteurs n'abordent pas le problème de la maintenance. Or, la cardinalité maximale d'un cluster est de k, par construction. Il n'est donc pas trivial d'estimer si un nœud peut rallier un clusterhead présent dans son voisinage. Il est dans ce cas nécessaire d'échanger un certain nombre de messages, alors que ce clusterhead peut être loin. De plus, il est également obligatoire pour un clusterhead de maintenir une liste de ses membres, ce qui introduit un nouveau surcoût.

[5] présente une approche similaire. La première étape construit un spanning tree. Ensuite, les clusters sont formés par branche, sous la direction de la racine de cette branche. Appelons A et B les deux branches en considération, et $|A|$ et $|B|$ leur cardinalité respective.

- Si $|A| + |B| < 2 \cdot k$ alors on fusionne les branches dans un même cluster ;
- Si $|A| + |B| < 2 \cdot k$, $|A| > k$ et $|B| > k$, A et B restent clusters *normaux* ;
- Si $|A| + |B| < 2 \cdot k$ et $|A| < k$, A constitue un cluster *partiel*.

Les clusters partiels seront ensuite fusionnés avec d'autres clusters partiels et la racine pour former un même cluster. Chaque sommet étant examiné itérativement à partir des feuilles, le processus converge pour clusteriser tout le réseau. Les auteurs montrent qu'il existe peu de clusters partiels lorsque le processus est fini, donc tous les clusters possèdent une taille comprise entre k et $2k$. Les auteurs proposent également une maintenance. Un nœud cherchant un clusterhead peut se raccrocher à un cluster dont la taille n'excède pas $3 \cdot k - 1$. S'il n'existe que des clusters candidats de taille trop importante, le nœud se rajoute à l'un d'eux et provoque la fission de ce cluster en deux sous-clusters de taille comprise entre k et $2k$. Si un nœud au contraire quitte le cluster, il se peut qu'il provoque une déconnexion. Le cluster se divise alors en fragments qui tenteront de fusionner avec d'autres si leur taille est inférieure à k. Les auteurs proposent donc des algorithmes de construction et de maintenance avec un nombre borné de participants, variant néanmoins du simple au triple. Un diamètre

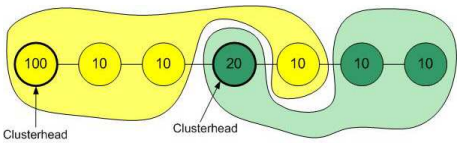

Figure 3: Non connexité des clusters sans l'optimisation de [4] avec un rayon de cluster de 5

borné nous paraît préférable car il permet de limiter le nombre de sauts pour acheminer le trafic de contrôle dans un cluster. De plus, les auteurs suppriment le concept de clusterhead, une élection parallèle étant nécessaire si un chef est requis. Enfin, un changement local de lien peut engendrer une modification brutale des clusters plus ou moins proches, avec une série de fissions et fusions de clusters.

Les auteurs de [4] ont une approche plus classique. Ils proposent l'élection d'un nœud de poids maximum dans son k-voisinage. L'élection est basée sur l'identifiant, mais l'algorithme peut être généralisé avec un poids quelconque. L'algorithme se décompose en 2 phases. Dans un premier temps, chacun diffuse son *gagnant*. Le *gagnant* est au début fixé à sa propre adresse puis il contient l'adresse du nœud de poids le plus fort entendu dans la première phase. Cette étape dure k rounds et propage les adresses des nœuds de poids le plus élevé à k sauts de distance. La deuxième phase sert au contraire à notifier aux clusterheads qu'ils ont été choisis par un nœud et donc qu'ils doivent les servir. Chaque nœud va donc propager les identifiants des nœuds de plus petit poids, entendus au dernier round de la première phase. Si un nœud entend son adresse dans la 2° phase, il se déclare clusterhead. Cependant, une telle méthode présente le désavantage de la création de clusters non obligatoirement connexes (figure 3), ce qui peut poser des problèmes dans certaines utilisations. Aussi, les auteurs proposent de choisir comme clusterhead le nœud de plus faible poids entendu dans la deuxième phase. Cependant, une telle méthode a pour conséquence de ne pas maximiser la cardinalité des clusters dont les clusterheads sont les plus forts, c'est à dire les plus aptes à jouer leur rôle. La complexité en temps donnée par les auteurs est en $O(2d + d) = O(d)$ (2d rounds et d sauts pour avertir les membres). Nous estimons d'autre part la complexité en messages à $O(n \cdot 2d \cdot \Delta)$, un message étant envoyé de façon fiable à chaque round par chaque nœud.

Les auteurs de [17, 26] utilisent l'algorithme classique de création de 1-clusters pour l'étendre aux k-clusters. Un leader décide de l'initialisation du clustering. Chacun ayant des informations sur ses k-voisins, il s'élit s'il possède le plus fort couple identifiant-degré puis l'annonce à son k-voisinage. Si un nœud intermédiaire n'a pas encore choisi de clusterhead, alors il le choisit. Un nœud ne diffuse sa décision que quand tous les nœuds d'identifiant-degré supérieurs l'ont fait. Un nœud peut être présent dans plusieurs clusters si plusieurs clusterheads sont à moins de k sauts, et deviennent nœuds bordure. Cependant, l'appartenance à plusieurs clusters nous semble multiplier la surcharge réseau quant à l'enregistrement de chaque membre auprès du clusterhead, et la vérification de connexité avec chacun des clusterheads. Les auteurs proposent une procédure de maintenance. Si un

nœud apparaît, il rejoint un cluster existant s'il le peut, sinon il en forme un nouveau. Si un clusterhead s'éteint ou qu'un lien utile casse, les nœuds restant élisent un ou plusieurs clusterheads (selon le nombre de nœuds qu'il couvre). Une telle élection n'est pas décrite mais les auteurs proposent en exemple la méthode de [36], dont nous avons déjà pointé les inconvénients. Cependant, les auteurs ne proposent pas de procédure de maintenance pour supprimer la redondance entre les clusters, dont ils repoussent l'étude à un futur article. Cependant, une telle procédure contrebalançant la création de clusters est obligatoire, sous peine d'avoir un nombre croissant de clusters. Ils ne proposent d'ailleurs pas dans leurs simulations d'étude temporelle de leur clusterisation.

1.3 Minimum Connected Dominating Set

1.3.1 Intérêt

Un *Minimum Connected Dominating Set* permet de modéliser un backbone, collectant le trafic. Un tel backbone permet de diffuser de façon efficace le trafic de contrôle au sein du réseau. Grâce à une telle structure, les problèmes de collision et de redondance de transmissions sont limités, et le nombre de participants est optimisé. Ensuite, nous pensons qu'un tel arbre nous permettra à terme d'optimiser la diffusion de trafic de contrôle de localisation. Les *advertisements* de Mobile IP [14] seront directement diffusés dans ce backbone, aidant à la maintenance, et permettant d'optimiser les handovers[10].

1.3.2 Définition

Un ensemble dominant total V' de V est tel que pour tout noeud n'appartenant pas à cet ensemble, il existe un voisin appartenant à cet ensemble dominant. Un ensemble de *clusterheads* est par exemple un ensemble dominant total.

$$\forall u \in V, \exists v \in V' \ / \ v' \in N_1(u) \tag{2}$$

Un *Minimum Connected Dominating Set* est un ensemble de nœuds dominants, tels que cet ensemble est connecté et de cardinalité minimale. Par extension, un k-MCDS est un ensemble dominant connecté tel que tout noeud possède dans son k-voisinage un dominant(fig.4).

$$\forall u \in V, \exists v \in V' \ / \ v' \in N_k(u) \tag{3}$$

1.3.3 1-MCDS

Construction La construction centralisée d'un MCDS est un problème NP-difficile. La construction décentralisée l'est par conséquent également. Le but de tout algorithme est

[10] un client changeant de points d'accès pour accéder au réseau filaire et à Internet

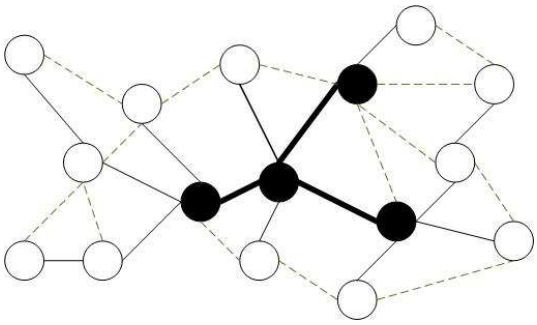

Figure 4: Exemple de 2-MCDS

donc de proposer la construction d'une structure approchant le plus possible d'un MCDS, la moins coûteuse possible en messages et en temps.

Les auteurs de [27] proposent deux algorithmes centralisés polynomiaux de construction de MCDS. Les auteurs introduisent deux couleurs : le noir pour les dominants, donc les membres du MCDS, le gris pour les dominés, et le blanc pour les nœuds dont l'état n'est pas encore déterminé. Ils construisent un arbre croissant en ajoutant à chaque étape un noir et en coloriant ses voisins en gris. Le 1^{er} algorithme colore le gris permettant la coloration du plus grand nombre de blancs. Le $2^{ème}$, lui, colore en noir la paire de nœuds qui permet la coloration du plus grand nombre de blancs. Les auteurs font remarquer que le $2^{ème}$ algorithme présente de meilleures performances dans beaucoup de configurations. Ces algorithmes centralisés présentent peu d'intérêt pour les réseaux ad hoc.

D'autres solutions créent un MCDS en deux étapes, en créant tout d'abord un ensemble dominant total puis en l'interconnectant. Il existe dans ce cas là 4 états possibles pour un nœud:

- dominant : un membre du MCDS ;
- dominé : un nœud *normal* qui possède un dominant voisin ;
- actif : un nœud en processus d'élection pour devenir dominant ;
- idle : état initial, dans lequel sont tous les nœuds lorsqu'ils apparaissent sur le réseau.

Dans [11, 13, 19, 35], un leader se déclare *dominant* et diffuse sa décision. Tous ses voisins deviennent *dominés*, et les voisins des dominés *actifs*. Ensuite, un actif est choisi dans le voisinage pour devenir dominant. Dans [11, 35], la métrique de décision est le degré, dans [13] l'adresse du nœud, dans [19] un coût de nœud (la valeur d'un tel coût n'étant pas spécifiée). La complexité en messages est donc en $O(n)$ (chacun donnant un seul message de décision) et celle en temps est en $O(n \cdot \Delta)$ (chaque paquet étant envoyé en broadcast).

L'interconnexion d'un tel ensemble n'est pas triviale. En effet, l'approche est relativement différente si on souhaite minimiser la cardinalité du MCDS, l'overhead de construction... Une première méthode, décrite dans [13], consiste à explorer chaque dominés pour savoir s'il serait un bon candidat pour l'interconnexion de dominants. Deux dominants sont éloignés de 2 sauts au maximum, puisqu'un dominé possède obligatoirement un voisin dominant. Les auteurs gardent la même symbolique des couleurs noirs/gris/blancs pour représenter les rôles des nœuds. Pour l'interconnexion, chaque dominé calcule son *degré noir*, c'est à dire le nombre de voisins noirs qui n'ont pas encore choisi de dominant ascendant, puis le diffuse quand la première étape est finie dans son voisinage. Chaque dominant ayant choisi un père va donc explorer son dominé à plus fort degré noir, celui ci devenant dominant et explorant chacun de ses voisins dominants sans père. Chaque nœud exploré va fixer l'expéditeur en tant que père. La complexité en messages est en $O(n)$ (un nœud étant exploré une seule fois) et en temps en $O(n)$ (puisque l'exploration est séquentielle). Il existe donc des délais croissants selon le nombre de nœuds. De plus, un tel algorithme est relativement peu adaptable à un k-MCDS : nous avons au maximum $2 \cdot k + 1$ sauts entre dominants, les dominés doivent donc également optimiser la longueur de connexion, information qu'ils ne possèdent pas. La complexité d'une telle exploration dans un environnement dynamique n'apporte donc pas de bénéfice certain par rapport à une connexion "au mieux". Les auteurs de [11] suivent une approche relativement semblable, en introduisant en plus un temporisateur, forçant un exploré à attendre que tous les nœuds de son voisinage deviennent soit noirs, soit gris. Un tel retard permet d'améliorer les performances de l'algorithme lors de l'exploration.

Dans [3], les auteurs proposent une méthode moins performante en terme de taille globale du MCDS, mais plus simple. Chaque dominant dont le voisinage ne comporte plus de nœuds *actifs* ou *idle* va diffuser un message d'invitation à la connexion, ce message étant relayé au maximum sur $2 \cdot k + 1$ sauts, distance maximale séparant deux dominants (fig. 5). Chaque nœud ne relaie qu'un seul message d'invitation à même TTL. Cette méthode permet de réduire les congestions sur le médium physique. Un dominant sans ancêtre dans le MCDS va fixer l'expéditeur du `invite` comme père et envoyer en unicast un `join` vers ce nouveau père. Chaque intermédiaire, dominé obligatoirement, deviendra dominant et relayera le message. Cependant, les auteurs n'ont effectué aucune simulation, aucune étude de l'évolution de ses propriétés au cours du temps... La complexité théorique de l'algorithme en messages est en $O(n)$ (chacun envoyant un seul message d'état dans la première étape, et k `joins` dans la deuxième étape), et une complexité en temps également en $O(n)$ (chaque nœud recevant une invitation, au pire de façon séquentielle).

[6] propose dans un premier temps de créer un ensemble de dominants, appelés clusterheads. Puis les auteurs proposent d'élire des *doorways* et *gateways* pour interconnecter l'ensemble et créer un *Connected Dominating Set*. Un nœud est élu *doorway* si :

- il est entre deux clusterheads C_1 et C_2, à 3 sauts de distance ;
- aucun de ses 1-voisins est sur une route plus courte entre C_1 et C_2;
- aucun de ses 1-voisins de plus forte priorité est sur une route entre C_1 et C_2;

- aucun de ses 1-voisins ne correspond à un clusterhead et se trouve sur une route de 3 sauts ou moins entre C_1 et C_2.

Un nœud est *gateway* si :

- il est entre deux clusterheads C_1 et C_2 éloignés l'un de l'autre de 2 sauts ;
- C_1 et C_2 ne diffusent aucun nœud soit clusterhead soit de plus forte priorité, unique intermédiaire entre C_1 et C_2.

Ces connaissances sont déduites des informations locales, chacun envoyant ses 1-voisins. Les auteurs proposent de donner une *priorité* à un nœud, dépendant de sa vitesse, de son énergie... Le même algorithme est utilisé pour la maintenance et la construction, les règles devant être toujours respectées. Les auteurs proposent une mise à jour périodique mais ne donnent pas la méthode de déclenchement de la mise à jour, sans doute parce qu'une synchronisation dans un réseau hybride est particulièrement complexe. Les règles données par les auteurs ne permettent pas de supprimer tous les cas de redondance des *doorways* et *gateways*, elles ne permettent que de les limiter. Enfin, nous pensons que l'adaptation de cet algorithme à la construction d'un k-MCDS peut engendrer des limitations. En effet, le nombre de chemins possibles va augmenter, augmentant ainsi la complexité des élections des *doorways* et *gateways*. De même, la redondance des *doorways* et *gateways* aura tendance à être plus importante.

Dans [19], les auteurs choisissent de propager dans la première étape un rang. Pour un dominé, le rang est le rang du leader. Pour un leader, le rang est le rang le plus élevé des voisins déjà dominés. Cette métrique représente donc une sorte de *distance* par rapport au leader. Un dominant peut donc choisir comme père le dominé qui possède le plus faible rang et qui interconnecte le plus de dominants. Celui de plus faible rang est obligatoirement plus proche de la racine du MCDS, donc itérativement, l'ensemble du MCDS se connecte à la racine. De même, un dominé peut optimiser son dominant en choisissant celui de plus faible rang entendu. Les MCDS dans les simulations des auteurs présentent une taille intéressante. La complexité en messages est en $O(n)$ (chacun envoyant un message de décision et un message de poids) et en temps en $O(n \cdot \Delta)$ (chaque broadcast prenant Δ temps). Néanmoins, la prise en compte seulement du degré ne garantit pas une bonne stabilité de la structure. Enfin, lorsque nous construisons un k-MCDS, tous les dominés d'un même dominant possèdent un même rang. Donc, un dominant n'a pas moyen de distinguer le dominé auquel il vaut mieux s'interconnecter pour optimiser la longueur du chemin.

La création de bases de données de localisation dans [35] se rapproche de la construction d'un k-MCDS. Les auteurs créent des bases de données formant un ensemble dominant total. Un nœud peut avoir trois états : *normal* (un dominant est à portée), *panique* (il est isolé), et *samaritain* (il possède un dominant mais aussi des voisins en mode *panique*). A chaque étape, au moins un nœud panique ou samaritain est élu pour devenir dominant. Un tel nœud est élu s'il possède le plus fort degré de nœuds paniques dans son voisinage. L'algorithme est réitéré jusqu'à que tous les nœuds soient couverts. Les bases de données

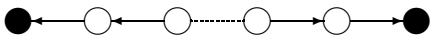

Figure 5: Distance maximale entre deux dominants dans un 2-MCDS

se trouvent à maximum 2k+1 sauts (figure 5). Les auteurs proposent donc que ces bases de données envoient des paquets `hellos`, avec un TTL de $2 \cdot k + 1$. Les bases de données peuvent donc établir leurs voisines logiques et implémenter n'importe quel algorithme de routage. Pour éviter les redondance inutiles, les bases de données situées à moins de D sauts fusionnent. Le problème est donc de régler D pour éviter une redondance importante tout en évitant d'introduire un overhead dû aux mises à jour incessantes des bases de données. Nous estimons la complexité en messages en $O(n^2 \cdot \Delta_k)$ (le processus requérant n étapes au maximum pendant lesquelles chacun envoie au maximum un message d'état et de nombre de voisins non couverts) et celle en temps en $O(n \cdot \Delta_k)$ (chaque étape demandant un paquet de broadcast). Il nous semble qu'une approche traduisant plusieurs critères tels que la mobilité, le degré, etc. pourrait conduire à une structure plus stable. De plus, il nous semble que la méthode de suppression de redondance de dominants présente des difficultés de réglage de seuil de fusion. En effet, dans un environnement statique, il serait intéressant de fusionner des bases lointaines, alors qu'un tel procédé est à éviter dans un environnement très volatile. Enfin, l'overhead de fusion peut être non négligeable. Il est en effet nécessaire de prévenir tous les anciens clients, et déplacer les informations de l'ancienne base à la nouvelle.

La connexion proposée dans [44] présente un concept intéressant pour l'interconnexion d'un MDS[11]. Un dominé va écouter les informations de topologie envoyées par ses voisins dominés et scruter son dominant pour savoir s'il retransmet ces informations. Si c'est le cas, alors il peut en déduire que le backbone est connecté. Sinon, il va devenir dominant pour permettre au backbone de se connecter. Un tel principe est possible puisque deux dominants dans un 1-MCDS sont séparés par un maximum de 2 dominés. Cependant, on ne peut créer ainsi un k-MCDS. De plus, ceci suppose que les informations de topologie sont obligatoirement inondées dans tout le backbone. Si les informations sont propagées dans un rayon limité comme le proposent les auteurs, il se peut que des dominés deviennent dominants inutilement (deux nœuds peuvent être géographiquement proches mais éloignés via le backbone).

Dans [48], les auteurs créent des dominants sans les interconnecter. Ils estiment l'interconnexion non robuste et utilisent donc un mécanisme de sélection de nœuds relais pour joindre les autres nœuds du cœur. Au sein de chaque dominant, les auteurs créent une sorte d'arbre local à partir de toutes les informations de voisinage et des paquets de *beacons* diffusés par les nœuds du cœur. Cependant, une telle solution requiert la diffusion d'un grand nombre de paquets dans le (2k+1)-voisinage, quelle que soit la mobilité des nœuds, diffusion

[11] *Minimum disconnected Dominating Set*

d'autant plus importante pour un k-MCDS avec k élevé. Le mécanisme de diffusion étant très coûteux, nous pensons qu'il est plus judicieux de les interconnecter et d'exécuter une maintenance purement locale. Les auteurs ne comparent d'ailleurs pas l'overhead de leur proposition avec celui d'autres solutions. [7] suit une approche similaire en créant un ensemble dominant dans une première étape, puis en interconnectant tous les dominants présentes à 2k+1 sauts (3 sauts dans leur exemple, puisqu'ils construisent un 1-MCDS).

Maintenance Comme nous l'avons vu, de nombreux algorithmes existent pour construire un MCDS au sein d'un réseau ad hoc. Cependant, la plus grande partie des algorithmes [3, 11, 13, 19, 27] ne s'attache qu'à minimiser la taille de l'arbre construit. Aucune étude de l'overhead réseau pratique en messages ou de la stabilité n'est réalisée, ce qui constitue pourtant des paramètres importants dans les réseaux ad hoc où la bande passante est limitée. De même ces algorithmes, à part [35], ne présentent aucune procédure de maintenance. La structure doit présenter une taille réduite au départ, mais il est plus utile en environnement dynamique de posséder des propriétés stables au cours du temps. La taille doit rester acceptable et la structure connectée pendant toute la maintenance, tout en introduisant un overhead minimum.

1.3.4 Applications

Cedar Cedar [47] présente une proposition très intéressante de routage sur un backbone MCDS. Comme dans beaucoup d'algorithmes, une première étape permet de créer un ensemble dominant total, un MDS. Un nœud choisit comme dominant celui qui possède le plus de dominés. S'il y a égalité entre deux nœuds, il choisit celui qui possède le plus fort degré, ce qui est le cas lors du début de la construction par exemple. Tout nœud qui possède au moins un dominé fait partie du cœur. Cette procédure est répétée périodiquement, ce qui sert de maintenance. Chaque dominé envoie à son dominant son voisinage ainsi que les dominants de ses voisins.

Le cœur doit ensuite se connecter. Chaque dominant diffuse donc une requête, chaque nœud relais rajoutant son adresse à la volée. La première étape construit un 1-MCDS, donc les dominants sont espacés de trois sauts au maximum, valeur du TTL du message. Un nœud du cœur recevant un tel message enregistre le lien virtuel l'unissant à ses voisins et la liste des intermédiaires nécessaires.

Pour la diffusion de la topologie, chaque nœud donne les changements de liens et de leur bande passante à son dominant. Ensuite, la diffusion d'une telle information incombe aux dominants. Les auteurs proposent un mécanisme de transmission de changement de topologie par vagues. L'information d'une augmentation de bande passante se déplace moins rapidement qu'une diminution. Ainsi, seuls les liens stables sont connus des nœuds lointains.

Lorsqu'un nœud A souhaite connaître une route vers un nœud B, il demande à son dominant D. Si D ne connaît pas le dominant de B, il diffuse une demande dans le cœur. Puis, la route est calculée grâce aux informations locales qui sont obligatoirement exactes et celles globales qui peuvent être périmées. Si D trouve une route complète, il envoie directement le paquet, sinon, il calcule la route vers un dominant D' plus proche de la destination et lui

envoie le paquet. Une route est calculée grâce aux informations de bande passante restante sur chaque lien, ce qui offre donc une certaine qualité de service si cette bande passante ne varie pas au cours du temps. Cependant, dans un environnement très dynamique tel que les réseaux ad hoc, une telle réservation peut se trouver fortement perturbée.

Les concepts de vague de propagation des changements, de proposition d'un routage avec QOS et chemin de secours par le cœur présentent des atouts intéressants et peuvent être réutilisables dans des solutions de routage sur backbone. Cependant, les simulations de cette solution présentent certaines limites : l'évolution du cœur en fonction de la vitesse des nœuds, le nombre de tunnels, les performances lors du départ d'un nœud du cœur, le temps de convergence de la construction du MDS n'ont pas été étudiés. De plus, Cedar choisit de créer des liens virtuels sur des liens physiques multisauts. Un dominant se déplaçant peut donc rendre un tunnel non optimal, allongeant inutilement le chemin. En outre, Cedar construit son MDS en ne prenant en compte que le degré, alors que d'autres critères semblent rentrer en compte pour la construction d'une structure stable. Enfin, l'adaptation de Cedar à un k-MDS semble poser des problèmes pour l'interconnexion des nœuds du cœur : la diffusion périodique de paquets de `piggybacked-broadcast` risque d'engendrer des collisions et engorger le réseau, tout en multipliant le nombre de tunnels.

[51] reprend le concept du *core* de Cedar, mais supprime la notion de lien virtuel entre nœuds du *core*. Les nœuds du *core* utilisent un *Forwarding Set*, comme dans OLSR, pour déterminer les relais à sélectionner pour toucher l'intégralité de sa zone et éventuellement des autres nœuds du *core*. Ce *Forwarding Set* est tiré des informations contenues dans les `hellos` des k-voisins, contenant leur identifiant, leur dominant, et le dominant de chacun de leur voisin. Le sous arbre de diffusion est calculé au sein de chaque nœud du *core*, ce qui rend la structure plus robuste. Cependant, les auteurs ont abandonné les autres concepts de Cedar et utilisent un tel *core* uniquement pour la diffusion des *Route Request* d'AODV et DSR.

Spine routing Les auteurs de [22, 49, 50] proposent une solution créant un backbone proche d'un MCDS. Un k-ensemble dominant total basé sur le degré, proche de [35], est tout d'abord construit. Puis, un réseau virtuel est créé en supprimant les liens entre dominés de même dominant, et en pondérant ceux entre dominants/dominés de telle sorte que les plus longs chemins entre dominants possèdent un poids élevé. Puis, ces dominants s'interconnectent en créant un spanning tree sur ce réseau virtuel.

En maintenance, si un dominé se déplace, il se réinscrira auprès d'un autre dominant, ou demandera à un dominé de devenir dominant s'il se retrouve hors de portée du backbone. Si un dominant, feuille du MCDS, se déplace, ses dominés chercheront un nouveau dominant et il est vraisemblable que l'un d'entre eux sera élu pour la succession. Si un nœud important du backbone se déplace, un algorithme de maintenance de spanning tree peut être exécuté. Ensuite, il est possible de diffuser sur le backbone toute la topologie, donc de créer un overhead très élevé. Mais les auteurs proposent également un mécanisme de vagues inspiré de Cedar.

Dans la première étape, un dominant au moins est élu à chaque round. Il est en effet possible que tous soient en interblocage et attendent l'élection d'un autre (ex. de la chaîne de nœuds de poids croissant). La complexité en temps de la première étape est donc en $O(|C| \cdot \Delta)$, avec C l'ensemble des dominants, chaque message étant envoyé de façon fiable au voisinage. Si nous utilisons l'algorithme de spanning tree cité en exemple par les auteurs, la complexité de la deuxième étape est en $O(n \cdot \Delta)$, n étant la cardinalité du graphe. Pour la complexité en messages, les auteurs donnent une complexité globale en $O(n \cdot |C| + m + n \cdot log(n))$, m étant le nombre d'arêtes du graphe.

Le *Spine Routing* propose donc un concept de backbone virtuel intéressant, et montre que différentes techniques de routage sont possibles sur une telle structure. Cependant, le MCDS créé ne prend en compte que le degré, critère limité pour traduire l'aptitude d'un nœud à devenir *maître*. De plus, l'interconnexion des dominants est un problème complexe. Les auteurs proposent d'utiliser un algorithme de spanning tree, reportant la complexité sur les algorithmes déjà existant. Le délai de construction du MCDS, même théorique, reste plus élevé. L'extension d'un tel algorithme à un k-MCDS semble poser des problèmes dans l'établissement du coût des liens pour le spanning tree. En effet, dans un k-MCDS, le lien entre deux dominés à même dominant peut être nécessaire pour l'interconnexion. L'algorithme de spanning tree possède donc un coût élevé sur une topologie aussi importante. Enfin, la maintenance est un élément important pour un MCDS, cependant, les auteurs détaillent peu leur maintenance et ne simulent que le mouvement d'un nœud dominé ou feuille du backbone, sans aborder le problème du mouvement d'un nœud intérieur du *spine*.

Backbones hétérogènes Certains réseaux ad hoc, comme ceux dédiés aux opérations militaires, sont très hétérogènes et logiquement hiérarchisés. Les solutions proposées sont donc spécifiques à de telles applications. Les auteurs de [2] proposent la création d'une topologie virtuelle avec des *Mobile Base Stations* (MBS) à capacité importante, possédant une capacité de communication longue portée, et des *Mobile Terminals* (MT) plus faibles, ne pouvant communiquer qu'avec l'aide des *MBS*. Les *MBS* accèdent au réseau filaire via des *Switching Center* (SC).

Dans un premier temps, les MBS sont découpés en zones selon le SC qui les dessert. Puis un arbre est construit pour chaque zone. Une MBS est choisie comme racine, puis à chaque étape, une MBS est ajoutée à l'arbre. Un tel réseau permet donc de reproduire le schéma des réseaux cellulaires, avec des *Access Points*, et des clients communicant via leur AP. Cependant, une MBS pouvant se déplacer, un roaming est possible, ses anciens clients se rattachant aux MBS voisines ou la suivant dans son déplacement. Les auteurs proposent également un algorithme de distribution de charge dans lequel les MBS sont déplacées selon une décision centralisée pour maximiser l'efficacité globale du réseau. Une MBS en sous-charge va être dirigée géographiquement vers des MBS en surcharge pour prendre en charge les clients en surnombre. Nous pensons donc qu'une telle approche est trop spécifique pour être efficace dans un réseau hybride classique, où tous les nœuds sont libres de leurs mouvements. Le concept de terminaux légers peut par contre être repris. De

plus, la création d'arbres de MBS n'est pas décrite en détails et le nombre de MBS par arbre est visiblement réduit, ce qui rend une adaptation aux réseaux hybrides difficiles.

[43] suit un peu même la logique avec l'élection d'un backbone parmi des nœuds à capacités élevées, les *Backbone Capable Nodes* (BCN). Ces nœuds sont chargés de couvrir des nœuds classiques, des *Regular Nodes* (RN). Les auteurs proposent des règles de conversion d'un BCN en membre du backbone (et vice versa) en fonction de RN à couvrir, et de la présence d'autres membres du backbone dans le voisinage. Cependant, il n'existe pas de preuve analytique d'exactitude de leur algorithme. D'autre part, les auteurs montrent grâce à des simulations que leur algorithme converge. Cependant, leur réseau étant peu dynamique, la topologie changeant à intervalles réguliers, toutes les 40 secondes, il n'existe aucune preuve de bon fonctionnement en environnement faiblement ou fortement dynamique de façon continue. Enfin, un tel algorithme n'est lui non plus pas très adapté à un k-MCDS, puisqu'il faudrait connaître son 2k-voisinage pour la sélection des nœuds du backbone.

Landmark Routing [58] propose la création d'un backbone, optimisant les performances globales du réseau. Cependant, un tel backbone ne doit pas constituer un *Single Point Of Failure*, le réseau pouvant continuer à fonctionner sans lui. Il existe une distinction entre les nœuds puissants, les *Backbone Nodes* (BN), et les nœuds normaux. Les BN ne pouvant être déployés de façon uniforme, il est nécessaire d'en sélectionner quelques uns afin de constituer le backbone. Les auteurs proposent donc un algorithme classique de clusterisation basé sur l'identifiant : le nœud à plus fort identifiant dans son voisinage s'élit clusterhead. Les auteurs précisent qu'un tel algorithme est adaptable pour faire du k-clustering, la connaissance du voisinage devant être diffusé à k sauts. Le backbone est ensuite constitué de l'ensemble de ces clusterheads, qui dialoguent directement grâce à un module radio longue distance, distinct de la radio utilisée pour les communications locales. Tout le trafic étant agrégé sur le backbone, son débit doit donc être largement supérieur à la bande passante dédiée aux transmissions courtes.

Pour le routage, les auteurs proposent de réutiliser le concept des Landmarks. Chaque groupe de nœuds (par exemple un bataillon de soldats) possède un Landmark (par exemple un véhicule blindé d'accompagnement), point de repère de la zone. A l'intérieur d'une zone Landmark, le routage est proactif car les communications y sont denses. La proposition initiale proposait d'utiliser le *Fisheye Routing* (FSR), mais tout protocole proactif convient. Ensuite, le routage entre Landmarks se fait grâce à une connaissance moins précise. Quand un nœud a une information à envoyer, il détermine le landmark destination, et tire de la table de landmark le prochain nœud sur la route. De proche en proche, le paquet atteint sa destination. Dans cette proposition, les auteurs utilisent le backbone pour accélérer les communications longue distance, mais il n'est pas nécessaire, le réseau pouvant jouer le même rôle, mais avec un nombre de sauts beaucoup plus élevé.

Cette proposition présente donc l'intérêt de l'utilisation d'un backbone pour le routage, pour la diffusion d'informations... Ce backbone est élu de façon dynamique avec un algorithme de clusterisation. Cependant, ses membres présentant des propriétés très particulières, une telle solution est très spécifiques aux applications militaires. Pour un réseau

hybride où tous les nœuds se déplacent de façon indépendante, où il n'existe pas deux modules radio différents pour les communications longues et courtes distance, l'adaptation d'une telle solution parait complexe. Nous pensons donc qu'une telle solution est peu adapté à un réseau hybride classique.

K-Tree Un *K-tree* est un arbre à k feuilles, minimisant la distance entre un nœud du réseau et un nœud de l'arbre. Les auteurs de [52] proposent dans un premier temps de créer un spaning tree dont les feuilles se trouvent à la périphérie du réseau. Les auteurs introduisent 3 couleurs : les noirs ont un père et un fils, les gris ont un fils mais pas de père, et les blancs n'ont rien du tout. La première étape, construisant une forêt, applique les règles suivantes :

- Si un blanc reçoit un message d'un de ses fils comme quoi il a été choisi comme père, il devient gris ;
- Si un nœud possède un cône d'ouverture α qui ne contient plus ni blanc ni gris, alors il choisit un voisin blanc ou gris comme père et se colorie en noir.

Il faut dans un deuxième temps interconnecter tous les arbres. Chaque racine envoie un identifiant d'arbre. Lorsqu'un nœud a reçu un identifiant de tous ses voisins, il choisit de relayer l'identifiant le plus fort, et si égalité, ayant parcouru le plus de sauts. Il choisit comme père l'émetteur de cet identifiant.

Puis, les auteurs proposent un algorithme permettant de trouver l'ensemble ϵ des k nœuds de l'arbre, permettant de minimiser la distance moyenne d'un nœud à ϵ. Les nœuds appartenant à ϵ seront les clusterheads de leur sous-arbre. Les nœuds de ϵ sont choisis comme ceux possédant la plus grande distance, appelée rang, vis à vis d'une feuille. Un clusterhead est en charge des informations de topologie intra-cluster et inter-clusters, de la découverte de routes le long de l'arbre vers les autres clusters...

Pour la maintenance, les auteurs proposent une série de règles en cas d'addition ou suppression de nœuds. Un nœud qui apparaît demande à se raccrocher à un de ses voisins. Ce nouveau père met à jour son rang, créant une mise à jour en cascade des rangs de ses supérieurs, et une mise à jour potentielle du clusterhead dont dépendait le nœud. Au contraire, un nœud qui disparaît entraîne la mise à jour du rang de l'ancien père, et la création d'orphelins, qui agissent comme s'ils venaient d'apparaître. Si un de ses orphelins ne trouve pas son père, il diffuse comme lors de la construction un message d'identifiant d'arbre, cherchant un autre arbre auquel se connecter.

Une telle solution est intéressante car elle élit des clusterheads permettant de minimiser la distance entre clusterheads et membres. Cependant, le nombre de clusterheads est fixe, ce qui défavorise l'adaptabilité au nombre global de nœuds dans le réseau. De plus, des antennes directionnelles sont obligatoires pour pouvoir départager un cône de réception d'angle α. La structure de k-arbre n'est pas construite selon des principes de stabilité, entraînant de nombreuses mises à jour, et des répercussions globales de changements locaux. Le trafic de contrôle est donc important, bien que la complexité en messages ne soit pas développée. De plus, il est possible que certaines oscillations globales existent dues à des oscillations locales.

Enfin, un clusterhead prend en charge toutes les fonctionnalités réseau, épuisant son énergie, alors qu'il n'a pas été élu selon ce critère. De plus, il doit choisir toutes les gateways vers les autres clusters, de façon centralisée.

Énergie [56] propose une suppression de redondance, qui peut être assimilée à un backbone. Un CDS[12] est intéressant pour la diffusion d'informations mais les membres consomment plus d'énergie. Il faut donc introduire des roulements pour que certains nœuds ne consomment pas plus leurs ressources que d'autres. Les auteurs proposent que certains nœuds s'endorment pendant un moment, pendant que les autres s'occupent du trafic de contrôle. Comme l'ensemble restant est obligatoirement connexe, il peut être assimilé à un backbone.

Les auteurs présentent des règles permettant à certains nœuds de s'endormir. Si un nœud N possède un voisin V dont le voisinage englobe son propre voisinage, et qu'il possède un identifiant plus petit, alors il peut s'endormir :

Soit $V \in N_1(N)$:

$$N_1(N) \subseteq N_1(V),\ id(N) < id(V) \tag{4}$$

La première condition correspond à une redondance de topologie, la deuxième à éviter que deux nœuds s'endorment simultanément s'ils possèdent le même voisinage. Les auteurs proposent quelques variantes plus élaborées afin de supprimer plus de redondance. Ils proposent par exemple de prendre en compte le degré et l'énergie plutôt que l'identifiant, et de comparer le voisinage du nœud N à deux de ses voisins, simultanément.

Soit $(V, W) \in (N_1(N))^2$:

$$N_1(N) \subseteq N_1(V) \cup N_1(V),\ \Delta(N) < \Delta(V)\ et\ \Delta(N) < \Delta(W) \tag{5}$$

$$N_1(V) \not\subseteq N_1(U) \cup N_1(W)\ et\ N_1(W) \not\subseteq N_1(U) \cup N_1(V) \tag{6}$$

L'ensemble de nœuds élus est mis à jour périodiquement. L'énergie des dominants diminuant, ils ne seront vraisemblablement plus élus la fois suivante, ce qui introduit une répartition de la charge entre les nœuds. Cependant, les auteurs ne détaillent pas la procédure permettant de synchroniser le début d'une telle élection, tâche non triviale dans un réseau ad hoc ou hybride. Les auteurs proposent donc une réduction de la redondance, mais sous estimée au vue de la restriction des règles. Les auteurs ne proposent pas de quantifier cette différence par rapport à l'optimal, ni analytiquement, ni par simulation. Une telle méthode peut présenter des performances moindres qu'un MCDS construit et maintenu. Enfin, les auteurs ne tiennent compte pour l'élection que de la réserve restante en énergie, la mobilité pouvant être ajoutée, selon nous, avec bénéfice.

WCDS Certains auteurs [18] introduisent la notion de *Weakly Connected Dominating Set*. Un *WCDS* est un ensemble de dominants connecté. Un WCDS s'approche donc des MCDS mais ne cherche pas obligatoirement à en minimiser la taille. Des boucles peuvent

[12] *Connected Dominating Set*

être présentes dans la structure, ce qui supprime la notion d'arbre. Une telle propriété introduit donc une augmentation du nombre de sauts nécessaires pour une inondation au sein d'un arbre, mais donne une certaine robustesse puisque la connexion du WCDS peut être redondante. Les auteurs proposent plusieurs variantes d'un algorithme centralisé, avec leur variante distribuée où la racine d'un spaning tree prend toutes les décisions de façon centralisée.

Les auteurs proposent également un algorithme distribué, avec la création d'un ensemble dominant dans une première étape. Les auteurs nomment *piece* un ensemble dans lequel tous les nœuds sont soit dominants soit dominés. Chaque *piece* possède un *piece id*, diffusé par un chef de zone. Initialement, chaque nœud possède un *piece id* différent: son propre identifiant. Les auteurs proposent de choisir comme dominant les nœuds qui permettent d'unir le plus de nœuds à *piece id* différent. Initialement, cette sélection est donc basée sur le degré, puis elle permet d'unir des *pieces* entre elles. L'élection est prise en charge pour chaque *piece* par le chef de zone : il envoie dans un premier temps une requête pour connaître tous les candidats à l'élection, en bordure de zone, puis il choisit le meilleur candidat qui devient dominant et nouveau chef de zone. Pour être sûr que les messages ne sortent pas de la *piece*, ils sont relayé seulement sur les liens dominant/dominé et dominant/dominant. Lorsque deux *pieces* fusionnent, le nouveau *piece id* est imposé par le chef de zone à plus faible *piece id*.

L'interconnexion des dominants est ensuite triviale. Comme deux dominants sont séparés par un dominé au maximum, il suffit d'autoriser le relais de paquets seulement sur les liens dominant/dominant ou dominant/dominé. Les auteurs stipulent que les *pieces* peuvent être initialement multiples. Mais ils ne détaillent que le processus de croissance des *pieces*, et pas celui de création. D'autre part, la centralisation de la décision au sein de chaque *piece* ne nous semble pas être une approche distribuée optimale. La prise de décision uniquement locale permet de réduire largement la complexité en temps et en messages, point non développé dans l'article pour l'algorithme distribué. Enfin, une telle construction se limite à un 1-WCDS, puisque sinon la connexion de l'ensemble est plus complexe.

2 Une topologie virtuelle adaptable

2.1 Découverte de Voisinage

Généralités La découverte de voisinage est un processus nécessaire à tout protocole de routage proactif [9, 10, 20, 21, 38]. Il faut échanger avec eux un certain nombre de paquets d'informations pour pouvoir, par leur intermédiaire, avoir une vue globale de la topologie. Le principe de découverte de voisinage est relativement simple : chaque nœud envoie à l'ensemble de ses voisins des paquets de `hellos`, contenant son identité. Cependant, les réseaux hybrides sont par essence même des environnements fortement dynamiques. Un rafraîchissement périodique de ces informations est donc nécessaire pour maintenir à jour le voisinage. Les paquets `hellos` sont donc envoyés toutes les I secondes, assez souvent pour que les informations soient les plus précises possibles, mais assez rarement pour ne pas

gaspiller de la bande passante radio. Il faut donc trouver un compromis et la bonne valeur de I. Si un paquet `hello` subit une collision et qu'il n'arrive pas, il ne faut pas pour autant supprimer immédiatement ce nœud du voisinage. D'un autre côté, un nœud qui est parti du voisinage, n'émettant plus d'`hellos`, doit être supprimé de la liste. Il faut donc introduire un *timeout* supérieur à I, mais relativement faible pour limiter les informations périmées. La connaissance de ce voisinage nous permet dans un protocole proactif d'ensuite échanger des paquets de topologie, donnant par exemple la position d'un nœud par rapport à ses voisins. De proche en proche, des routes sont connues par tous les nœuds.

Format Les protocoles réactifs, tels DSR [32], ne requièrent pas de découverte de voisinage. Ils agissent à la demande. Lors de l'envoi d'un paquet, la source émet une `Route Request`, et reçoit en réponse une liste d'intermédiaires pour atteindre la destination. Cependant, lors de l'utilisation d'une telle route, il n'existe aucun mécanisme permettant de détecter la cassure de façon proactive. L'absence d'acquittements signifie que la route est cassée. Il faut alors notifier la source, qui doit stopper son émission et redécouvrir une nouvelle route. Les délais sont donc importants, alors que la proactivité permet d'introduire plus de services avec la détection proactive de cassure de routes, la reconstruction locale de routes.

Nous avons donc choisi d'utiliser des paquets `hellos` classiques véhiculant les informations suivantes :

- Source : émetteur ;
- Destination : nœud destinataire, pour un paquet `hello`, $destination = -1$, ce qui correspond à un broadcast ;
- Poids : poids de l'émetteur ;
- Identifiant : Identifiant du paquet (pour détecter les paquets en doublon sur le réseau) ;
- TTL : champs permettant de limiter la durée de vie d'un paquet en nombre de sauts ;
- État : Dominant, Dominé, Actif, ou Idle ;
- Relais : Précédent nœud ayant relayé le paquet ;
- Père : Père dans le MCDS de l'émetteur;
- Clusterhead : Clusterhead de l'émetteur.

Nous avons donc ajouté certains champs par rapport à un paquet `hello` classique, nous servant tant pour la construction du MCDS que celle des clusters. En effet, la contention pour l'accès au médium physique représente une part appréciable de la consommation de bande passante. Il est donc beaucoup plus économique sur un médium radio de fusionner plusieurs informations dans un seul paquet que d'envoyer plusieurs paquets de taille réduite. L'overhead que nous introduisons apparaît acceptable.

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Source							
Destination							
Type			Poids		Identifiant		
TTL	Etat						
Relais							
Père							
Clusterhead							

Table 1: Format du paquet hello de type relais

Nous avons également prévu un mécanisme de `fast-hellos`, à l'initialisation du réseau. Dans un tel état, les nœuds ne sont pas capables de communiquer avant d'avoir construit leur topologie virtuelle. Il est donc important de minimiser les délais de construction en envoyant des paquets `hellos` plus fréquemment. De même, nous utilisons également les paquets de `fast-hellos` pour intégrer plus rapidement les nouveaux nœuds apparaissant sur le réseau en cours d'utilisation. L'envoi de `fast-hellos` est donc enclenché dés qu'un voisin d'état 0 (initialisation) est détecté.

Méthodes Nous souhaitons construire des k-MCDS. La connaissance du k-voisinage est donc nécessaire. Nous proposons deux méthodes, complémentaires, utilisées dans des environnements différents:

- Relais : un paquet `hello` va être relayé dans un périmètre de k sauts autour de l'émetteur (tab. 1). Le champ TTL est donc utilisé dans ce but. Chaque nœud recevant un tel paquet va donc ajouter l'expéditeur dans sa table de voisinage, avec toutes les informations associées, et relayer le paquet en décrémentant le TTL, si toutefois ce champ n'atteint pas la valeur nulle.

  Cette méthode a l'inconvénient de créer de nombreux petits paquets, puisque chaque nœud émet un paquet tous les intervall_{hello} secondes. En moyenne, il existe donc un débit de $\Delta_k.intervall_{hello}$ paquets par seconde. Cependant, la convergence est extrêmement rapide pusiqu'il suffit d'attendre un intervall_{hello} pour connaître l'intégralité de son voisinage.

- Table : un paquet `hello` dans cette méthode contient la liste des (k-1)-voisins de l'émetteur (tab. 2). Les paquets sont diffusés aux 1-voisins, mais non relayés. De proche en proche, la liste des voisins étant diffusés, chaque nœud connaît ses k-voisins.

  Cette méthode permet de réduire le nombre de paquets diffusés. En moyenne, chaque nœud reçoit $\Delta_1.intervall_{hello}$ paquets par seconde, les paquets étant de taille plus importante. Cependant, le délai de convergence est plus élevé. Il faut $k \cdot intervall_{hello}$ paquets pour qu'un changement soit répercuté dans tout le k-voisinage.

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Source							
Destination							
Type		Poids		Identifiant			
TTL	Etat						
Père							
Clusterhead							
Adresse 1							
Poids 1		Sauts 1		Etat 1			
Adresse 2							
Poids 2		Sauts 2		Etat 2			
Adresse 3							
Poids 3		Sauts 3		Etat 3			
...							

Table 2: Format du paquet de type table

Ces deux méthodes peuvent donc être utilisées dans des cas différents. La méthode du *relais* sera donc plutôt utilisée en environnement fortement dynamique et lors de la période d'initialisation. Au contraire, la méthode de *table* sera utilisée lorsque le réseau se sera stabilisé. Il est par exemple possible de commuter dynamiquement sur une méthode ou l'autre en surveillant les changements dans la table de voisinage, avec un déclenchement sous forme de seuil d'hystérésis.

Liens unidirectionnels Dans un réseau hybride, il peut exister des liens unidirectionnels quand par exemple deux nœuds possèdent une portée différente. Peu de protocoles peuvent utiliser de tels liens, le mécanisme asymétrique étant complexe à gérer. Nous avons choisi personnellement de ne pas utiliser de tels liens. Cependant, il est nécessaire de les détecter. Lorsque les `hellos tables` sont utilisés, il suffit de vérifier que le voisin qu'on entend nous déclare dans sa table de voisinage. Lorsque les `hellos relais` sont utilisés, il faut ajouter la liste des 1-voisins pour pouvoir détecter les liens unidirectionnels.

2.2 Métrique

Nous souhaitons créer une structure virtuelle la plus stable possible. Il est donc nécessaire que les élections permettent de choisir des nœuds stables pour le backbone et les clusterheads. Nous avons donc choisi d'introduire une métrique de stabilité, rendant compte le plus fidèlement possible de la stabilité d'un nœud.

Cette métrique de stabilité dépend de plusieurs critères :

1. Persistance : un nœud N qui a été choisi comme membre du backbone ou clusterhead doit garder le plus longtemps possible son rôle. Le ratio du nombre de k-

voisins ayant choisi N comme dominant sur le nombre de ses k-voisins est représentatif de l'importance d'un dominant. De même, le ratio du nombre de k-voisins ayant choisi N comme clusterhead sur le nombre de ses k-voisins est représentatif de l'importance d'un clusterhead. Nous ajoutons ces deux métriques pour donner une métrique d'importance, forçant la stabilité de la structure virtuelle.

2. Mobilité : seule la mobilité relative nous intéresse ici. Il est inutile d'élire un nœud possédant une mobilité absolue faible alors que ses voisins changent beaucoup : il ne peut pas assumer sa tâche de façon efficace, ses clients étant très dynamiques. Nous avons donc choisi de mesurer la mobilité relative, et plus particulièrement les changements dans le voisinage, seul impact qui nous intéresse. Notre mobilité relative est mesurée par le nombre de voisins ayant changé au cours des n derniers intervalles sur le degré moyen durant ces intervalles, pour ne pas pénaliser les nœuds à forte densité.

3. Énergie : il est important qu'un nœud élu possède une énergie suffisante pour pouvoir assurer son rôle pendant un laps de temps suffisant. D'autre part, un dominant ou clusterhead consommera plus de ressources qu'un nœud normal, il faut donc pour optimiser la durée de vie des nœuds élire ceux à réserve plus élevée. Nous pensons qu'il faut pénaliser fortement les nœuds à faibles ressources. La métrique d'énergie devra donc diminuer de façon exponentielle par exemple.

4. Degré : un dominant/clusterhead doit prendre en charge suffisamment de clients pour que les dominants/clusterheads ne soient pas trop nombreux. Cependant, il faut également que ce nombre ne soit pas trop élevé pour éviter les congestions. Nous avons donc choisi de représenter la métrique de densité par la différence avec un degré idéal, correspondant à une congestion réduite, à adapter selon la densité moyenne de l'environnement. Nous pensons qu'un nombre de 10 voisins représente un bon compromis pour éviter les congestions. Au cours de nos simulations, nous nous sommes aperçus que les performances chutaient pour des valeurs plus élevées, phénomène que nous avons attribué aux collisions.

Nous normalisons toutes ces métriques entre 0 et 1 pour ne pas fausser l'influence de chaque critère. Puis nous obtenons une métrique globale de stabilité :

$$p_{global} = \alpha \cdot p_{persistance} + \beta \cdot p_{mobilite} + \delta \cdot p_{energie} + \gamma \cdot p_{densite} \qquad avec\ \alpha \gg \beta \gg \delta \gg \gamma \quad (7)$$

Cependant, nous pensons qu'à terme, une métrique non linéaire devrait être profitable pour privilégier les nœuds globalement bons, donner des critères éliminatoires (énergie trop faible, mobilité trop élevée)... Une métrique multiplicative pourrait donc par exemple être préférable, il serait intéressant d'en étudier le comportement dans de futurs travaux.

2.2.1 Acquittement

Les acquittements en unicast sont très simples : le destinataire envoie à l'émetteur un `ack`, confirmant la bonne réception du paquet. Cependant, les acquittements de broadcast sont

à bannir. En effet, dans un tel cas, l'intégralité des voisins doivent acquitter une émission. De plus, le paquet étant relayé pour la diffusion, chaque nœud acquitte l'émission de chacun de ses voisins, ce qui crée donc Δ acquittements, Δ étant la densité.

Nous avons donc choisi pour maximiser les chances de réception d'un paquet s'il est extrêmement important pour le fonctionnement du réseau, de le retransmettre $\max_{retry}$ fois, espacées de $\text{intervall}_{retransmit}$ secondes.

2.3 Backbone

Nous proposons de créer une approximation d'un k-MCDS, suivie d'une clusterisation. Cette structure virtuelle nous permet d'organiser notre réseau afin d'ajouter de nouvelles fonctionnalités et faciliter à l'avenir le déploiement de nouveaux services. Ces structures doivent parfaitement se combiner pour former un tout complémentaire, tout en limitant l'overhead de construction. Enfin, l'environnement d'un réseau hybride étant fortement dynamique, nous proposons une procédure de maintenance accompagnant la construction de nos 2 structures.

2.3.1 Intérêts

Une dorsale virtuelle Les réseaux cellulaires classiques utilisent un backbone filaire pour la collecte de trafic venant des liens radios terminaux. En construisant un backbone, nous créons une dorsale virtuelle pouvant être utilisée pour la diffusion. En effet, la diffusion d'une information de contrôle à tous les membres du réseau est une problématique importante dans un réseau hybride [39]. Il existe des problèmes de redondance de transmission, de congestions,... D'autre part, les paquets de broadcast n'étant pas acquittés, sous peine de congestion due à la multiplication de petits paquets, la fiabilité est très faible pour une diffusion, beaucoup de nœuds ne la recevant jamais à cause de collisions.

La mise en place d'une dorsale virtuelle permet d'introduire une diffusion à l'intérieur du backbone à l'aide d'unicast acquittés, ce qui augmente la fiabilité. D'autre part, la diffusion de beaucoup de paquets de contrôle pourra être, à terme, limitée aux membres du backbone, les informations n'étant relayés vers les nœuds *normaux* que s'ils le demandent. Ainsi, le backbone permettrait de supprimer une partir de la redondance des transmissions.

Gestion de la mobilité Les clients choisissent comme membre du backbone un nœud stable, qui puisse rester longtemps à portée, et donc qui puisse délivrer un service. Les membres du backbone peuvent donc jouer le rôle de *stations de base virtuelles*. Il serait intéressant de développer à terme une solution de gestion de la localisation des clients dans le réseau, adressables par le membre du backbone qui les dessert. Une solution de *paging* pourrait également être mise en place, le membre du backbone bufferisant les paquets à destination d'un de ses clients endormis. Le paging proposerait une solution d'économie d'énergie pour les clients légers ou possédant une faible réserve d'énergie.

Économie d'énergie Les nœuds du backbone dépenseront plus d'énergie puisqu'ils relayeront les paquets de diffusion dans le backbone, qu'ils prendront en charge la découverte et

le maintien des routes... Cependant, ces nœuds posséderont, de par leur élection, une réserve en énergie élevée. D'autre part, les nœuds normaux dépenseront moins d'énergie dans le contrôle. Ainsi, des nœuds à faible réserve d'énergie verront leur durée de vie augmentée.

Le backbone crée une hiérarchie dans le réseau, avec des nœuds puissants comme membres. Il est donc concevable d'introduire dans un réseau hybride des terminaux légers, tels qu'un PDA ou un téléphone cellulaire hybride. Ils choisiront un membre du backbone, se déchargeant sur lui d'une grande partie de leurs obligations.

2.3.2 Construction

Initiation Comme beaucoup d'algorithmes de construction de MCDS, nous possédons un leader chargé d'initier la construction du backbone. Il est naturel dans un réseau hybride de choisir comme leader l'AP. Il joue le rôle de passerelle pour accéder à Internet, une partie non négligeable du trafic passe donc par son intermédiaire. De plus, l'AP se plaçant à la racine du MCDS, il pourra diffuser plus facilement et efficacement les informations liées à la gestion de la mobilité, en diffusant par exemple les paquets Mobile IP. Enfin, à terme, chaque AP construira son propre MCDS, ce qui permettra de créer des zones de couverture de chaque AP. Un nœud choisira un membre du backbone, correspondant à un MCDS, et donc à un AP particulier, qu'il utilisera comme passerelle. Un nœud pourra donc détecter automatiquement le changement d'AP en observant les informations venant du MCDS. Il est donc logique que l'AP joue le rôle de déclencheur pour la construction du k_{mcds}-MCDS.

Découverte de voisinage Nous construisons notre k_{mcds}-MCDS en fonction de la connaissance de notre k_{mcds}-voisinage. Chacun prend une décision locale en fonction des informations qu'il possède sur ses k_{mcds}-voisins. Cependant, la connaissance d'un k_{voisin}-voisinage possède un coût : plus ce k_{voisin} est élevé, plus le coût est important. Les paquets de `hellos relais` sont plus nombreux ou les paquets `hellos table` sont de taille croissante. Cependant, nous devons au moins connaître notre k_{mcds}-voisinage, nous avons donc pour limiter au maximum l'overhead: $k_{mcds} = k_{voisin}$. Nous verrons plus loin que pour la construction et la maintenance, une telle valeur suffit.

Pour pouvoir construire le k_{mcds}-MCDS, la connaissance du k_{mcds}-voisinage doit être exacte pour chaque nœud. Il est donc nécessaire d'attendre le délai de convergence, d, des tables de voisinage. Il faut donc dans un premier temps estimer le délai maximum de traitement d'un paquet dans un nœud. Nous avons donc simulé une chaîne de 3 nœuds $1 \rightarrow 2 \rightarrow 3$, où le routage a été fixé de façon statique de 1 à 2, et de 2 à 3. Le nœud 1 envoie des paquets à débit constant vers 3. Nous avons augmenté progressivement la charge et avons mesuré au sein du nœud 2 le délai de *relais*, délai entre la réception de 2 et la réception de 3. Nous avons obtenu un délai de traitement maximum de 50 millisecondes. En réalité, le temps de traitement devrait dépendre d'autres facteurs comme le degré, la charge CPU induite par le traitement... Nous jugeons cependant qu'un tel temps représente une première approximation du temps de traitement maximum qu'un nœud peut introduire lors d'un relais de paquet.

A l'initialisation, seuls les paquets `hellos relais` sont utilisés. Il faut donc que le paquet soit propagé à k_{mcds} sauts avant que tout le k_{mcds}-voisinage l'ait réceptionné. Cependant, il a pu se produire des collisions. Nous considérons qu'un maximum de max_{retry} est possible. Ainsi, le délai avant la construction est donc de :

$$d = max_{retry} \cdot interval_{fast-hello} + k_{mcds} \cdot 0.05 \approx max_{retry} \cdot interval_{fast-hello} \quad (8)$$

Construction d'un ensemble dominant

Algorithme Nous avons choisi de créer dans un premier temps un ensemble dominant total. Les membres de cet ensemble seront choisi pour leur stabilité, c'est-à-dire leur poids élevé. Il existe pour un nœud 4 états, définissant son rôle :

1. Dominant (1) : membre de l'ensemble dominant, et donc membre du backbone ;
2. Dominé (2) : nœud extérieur au backbone, ayant choisi un dominant à moins de k_{mcds} sauts, jouant le rôle de père dans le k-MCDS ;
3. Actif (3) : nœud en processus d'élection pour devenir dominant ;
4. Idle (0) : nœud en état d'initialisation, attendant une sollicitation extérieur pour déterminer son état.

A l'initialisation, le leader, c'est à dire l'AP, va se déclarer dominant, donc membre du backbone. Chaque nœud changeant d'état va envoyer automatiquement un message d'état, pour que ses k_{mcds}-voisins changent les informations le concernant, et déclencher en cascade des décisions. Les règles suivantes sont appliquées (fig. 6), quelque soit le nœud :

1. Un nœud *idle* recevant un message d'état d'un *dominant* devient directement *dominé*, et choisit l'émetteur de ce message comme père ;
2. Un nœud *idle* recevant un message d'état d'un *dominé* devient *actif* ;
3. Un nœud actif possédant le poids le plus élevé dans son k_{mcds}-voisinage de nœuds actifs, pendant τ temps, devient *dominant*. Il ne possède pas encore de père durant cette phase, l'interconnexion étant réalisé durant la deuxième phase ;
4. Un nœud actif recevant un message d'état d'un dominant devient dominé et choisit l'émetteur comme père.

En résumé, l'algorithme a le comportement suivant : le leader devient *dominant*, et envoie donc un message d'état. Tous ses k_{mcds}-voisins recevant un tel message vont donc devenir directement ses dominés. Ils vont déclarer leur nouvel état à leurs voisins. Les k_{mcds}-voisins de ces dominés vont donc devenir eux *actifs*, rentrant dans le processus d'élection. Au bout d'un temps τ, au moins un de ces *actifs* va être élu dominant, parce qu'il possédera le plus

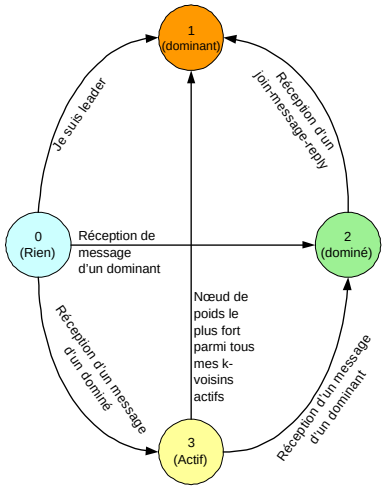

Figure 6: Diagramme de changement d'état pour la première phase de construction du k-MCDS

fort poids parmi tous ses k_{mcds}-voisins actifs. Ce nouveau *dominant* va forcer ses k_{mcds}-voisins à passer de l'état *actif* ou *idle* à l'état *dominé*. Et le processus réitère, les *dominés* entraînant de nouveaux *actifs*. Le processus agit donc quasiment en cercle, un *dominant* entraînant une première vague de *dominés*, qui entraînent eux mêmes une deuxième vague d'*actifs*, parmi lesquels un ou plusieurs nœuds seront élus *dominants*. Le processus se déroule donc en 2 grandes étapes successives.

Il est important que les messages d'état des *dominants* soient relayés en priorité par rapport aux messages d'état des *dominés*, pour forcer le plus possible de nœuds à devenir dominés, sans passer inutilement par l'état actif.

Chaque nœud envoie donc au maximum deux messages de changement d'état, d'où :

$$C_{messages} = O(n) \tag{9}$$

D'autre part, il existe à chaque round l'élection d'au moins un dominant dans un voisinage de nœuds actifs. Un nouveau dominant est espacé d'au moins $k_{mcds}+1$ sauts d'un autre dominant. Le processus d'élection se déroule sous forme de vagues, centrées sur le leader, chaque élection prenant τ temps. La complexité en temps est donc, si Θ est le diamètre du réseau et d le délai de propagation d'un message d'état sur un saut:

$$C_{temps} = O\left(\frac{\Theta}{k_{mcds}+1} \cdot (\tau + d \cdot k_{mcds})\right) \tag{10}$$

$$= O\left(\frac{\Theta}{k_{mcds}+1} \cdot \tau\right) \tag{11}$$

Temporisateur Lorsqu'un nœud N est actif et qu'il possède un poids maximum par rapport à tous ses autres k_{mcds}-voisins actifs, il doit obligatoirement attendre un temps τ. En effet, il est nécessaire pour minimiser le nombre de dominants élus de laisser à tous les nœuds du k_{mcds}-voisinage le temps de se déclarer actifs. Le paquet de changement d'état ayant provoqué l'activité de N peut encore se propager au maximum à k_{mcds}-1 sauts. Ensuite, ce nœud éloigné doit avoir le temps de transmettre lui aussi son nouvel état, à k_{mcds}-1 sauts également. Enfin, un paquet ayant été retransmis max_{retry} fois est considéré comme perdu (section 2.2.1 p.31). D'où :

$$\tau = 2 \cdot (k_{mcds} - 1) \cdot (0.05 + timeout_{retry} \cdot (max_{retry} - 1)) \quad (12)$$

Connexion du backbone La première phase a permis de créer un ensemble dominant total. Il faut dans une deuxième étape l'interconnecter pour créer un véritable backbone et approcher un k-MCDS. Cependant, il nous semble qu'une exploration itérative de chaque sommet est peu adaptée à un environnement dynamique. En effet, une telle méthode permet de minimiser le nombre de nœuds du k-MCDS mais présente des délais de convergence élevés. De plus, il est inutile de dépenser de nombreuses ressources pour la construction alors que l'environnement étant fortement dynamique, il est en perpétuelle mouvance, et donc que ses caractéristiques changent au cours du temps. Enfin, ces méthodes optimisent la taille en ne se basant que sur le degré, alors que la stabilité nous semble plus intéressante.

Deux dominants sont au maximum éloignés l'un de l'autre de $2k_{mcds}+1$ sauts, de par la définition d'un k-MCDS. Un dominant, lorsqu'il est lui même connecté peut donc inviter les autres à se connecter par son intermédiaire en envoyant un `join-message` (tab. 3) dont le TTL est fixé à $2k_{mcds}+1$. Au début de la construction, seul le leader est connecté, il envoie donc un `join-message` à ses $(2k_{mcds}+1)$-voisins. Ces voisins ne relaient le paquet que si leur état est soit *dominé*, soit *dominant*. Les nœuds *actifs* et *idle* stockent le message jusqu'à leur changement d'état. Une telle restriction permet de forcer la fin de la première étape de notre algorithme avant le commencement de la deuxième, tout en minimisant les délais de construction. Les dominés vont donc relayer un `join-message` en y ajoutant leur adresse. Un dominant non connecté recevant ce paquet va armer un temporisateur, pour stocker de possibles futurs autres `join-messages` et pouvoir choisir le dominant le plus stable auquel se connecter. Au bout de ce temps, il choisit le meilleur candidat, envoie un `join-reply` suivant le chemin inverse et forçant tous les dominés intermédiaires à devenir dominant. Le nouveau dominant connecté peut donc lui aussi envoyer un `join-message`. De proche en proche, l'ensemble du backbone va se connecter.

Cependant, chaque dominant connecté envoyant un `join-message`, il se peut qu'une inondation se créé. Nous proposons donc d'utiliser le mécanisme de [3] limitant la retransmission de nos paquets. Un dominé relayant un `join-message` avec un TTL de t, ne relayera plus d'autre `join-messages` avec un TTL inférieur ou égal à t, les dominants à cette distance ayant déjà pu se connecter.

Nous avons choisi dans notre structure de garder un ensemble d'informations permettant de parcourir le MCDS vers la racine et vers les feuilles. Un dominé garde l'identifiant du dominant auquel il se raccroche, que nous appellerons père. Un tel père est à maximum

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Source							
Destination							
Relais							
Type		TTL	Identifiant				
Relais1							
Relais2							
Relais3							
...							

Table 3: Format du `join-message`

k_{mcds} sauts. De même, un dominant garde en mémoire l'identité de tous ses dominés, qu'il tire de leurs `hellos` ou de leur message de changement d'état. Un dominant garde également l'identité de son père. Pour un dominé sur le chemin d'un `join-reply`, devenant dominant, c'est le nœud suivant sur le chemin du `join-reply`. Pour un dominant essayant de se connecter, envoyant un `join-reply`, c'est également le prochain nœud sur le chemin du `join-reply`. Garder en mémoire le père permet de posséder un chemin unicast vers la racine. De même, un dominant va garder l'identité de ses fils, c'est à dire les dominants l'ayant choisi comme père. Il connaît leur identité puisqu'ils lui font suivre un `join-reply`.

Un dominé relaie dans cette étape au maximum k_{mcds} `join-messages` à TTL différents. De plus, chaque dominant envoie au maximum un `join-reply` sur $k_{mcds}+1$ sauts et un `join-message`. La complexité en message est donc, si D est le nombre de dominants :

$$C_{messages} = O\left((n-D)\cdot k_{mcds} + D\cdot((k_{mcds}+1)+1)\right) \quad (13)$$
$$= O\left(n\cdot k_{mcds}\right) \quad (14)$$

Un dominant envoie un `join-message`, inondé dans le $2k_{mcds}+1$ voisinage. Le dominant le plus proche pouvant se connecter est à $k_{mcds}+1$ sauts. Le processus de connexion agit donc par vagues centrées sur le leader. La complexité en temps est donc, si Θ est le diamètre du réseau, d_{rep} la durée au bout de laquelle un dominant répond à un `join-message`, et d_{propag} le temps de propagation d'un `join-message` et `join-reply` sur un saut :

$$C_{temps} = O\left(\frac{\Theta}{k_{mcds}+1}\cdot(d_{rep}+2\cdot d_{propag}\cdot 2\cdot(k_{mcds}+1))\right) \quad (15)$$
$$= O\left(\frac{\Theta}{k_{mcds}+1}\cdot k_{mcds}\right) \quad (16)$$
$$= O\left(\Theta\right) \quad (17)$$

2.3.3 Maintenance

La maintenance est une problématique rarement abordé par les articles construisant un MCDS. Cependant maintenir la structure, pour qu'elle garde ses propriétés au cours du temps, est vital dans un environnement dynamique. Il est nécessaire que le MCDS reste connecté, et que chaque dominé continue à posséder un dominant présent à moins de k_{mcds} sauts. Il est important de noter que la construction est un phénomène rare, se produisant une fois à l'initialisation, alors que la maintenance est constamment active et donc doit présenter de bonnes performances. Reconstruire périodiquement la structure ne nous paraît pas une approche satisfaisante étant donné les délais induits, l'overhead,... Nous proposons donc une solution de maintenance complète, présentant de bon résultats au cours du temps, comme nous le montrons dans nos simulations.

Informations de descendance/ascendance Un nœud inscrit dans son paquet `hello` un certain nombre d'informations propres au MCDS telles que le père, l'état, le poids... Il est donc aisé de maintenir à jour la liste de ses dominés et de ses fils, en tirant ces informations des paquets `hellos`. Les informations sont donc continuellement réactualisées pour ne pas être périmées.

Relais vers le dominant Il se peut qu'un dominé ait besoin de dialoguer avec son dominant. Il doit donc posséder une route vers celui-ci. Le dominant envoyant à sa zone, régulièrement, des paquets de contrôle ou des paquets *hellos*, il suffit de noter l'identité du relais précédent pour connaître l'intermédiaire à contacter, sur le chemin de notre dominant.

Détection de déconnexion Le k-MCDS doit absolument rester connecté au cours du temps. Il est donc nécessaire de posséder un mécanisme permettant de détecter la déconnexion d'une branche du MCDS. Dans ce but, le leader, c'est à dire l'AP, envoie de façon périodique un `ap-hello` (tab. 4). Ce paquet va être relayé par les membres du k-MCDS. Un dominant recevant donc un `ap-hello` va donc vérifier sa connexité vis à vis de la racine du MCDS. Un dominant ayant raté plus de 2 `ap-hellos` va se considérer comme déconnecté. De même, si le père d'un dominant n'est plus voisin, alors le dominant se considère comme déconnecté. Nous avons donc tendance à toujours surestimer la déconnexion au backbone. Cependant, un dominant tentant de se reconnecter pourra stopper sa reconnexion s'il s'aperçoit qu'il est en fait toujours connecté.

Pour éviter une inondation des `ap-hellos` dans tout le réseau, ils ne sont pas distribués aux dominés. Les dominés surveillent juste le lien qui les unit à leur dominant. Ils possèdent une vision locale, laissant certaines fonctionnalités à leur dominant. Un dominé peut vérifier que le lien vers son dominant existe toujours en utilisant les paquets `hellos`, mais à terme il pourra utiliser les paquets de données et de contrôle en provenance de son dominant. Le dominant étant local, il devrait envoyer de nombreux paquets dans sa zone, ce qui devrait faciliter la détection de déconnexion.

Nous avons distingué 3 cas pour la maintenance, que nous traitons séparément :

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Source							
Destination							
Relais							
Type		TTL	Identifiant				
Poids		Etat					
Clusterhead							
Père							

Table 4: Format du `ap-hello`

1. Un dominé ayant perdu son dominant ;
2. Un dominant déconnecté tentant de se reconnecter ;
3. Un dominant ayant échoué sa reconnexion.

1-Maintenance des dominés Un dominé sans père va tenter d'en retrouver un avec les informations qu'il possède déjà. Il cherche dans son k_{mcds}-voisinage un dominant. S'il en existe un, il le choisit comme nouveau père. S'il en existe plusieurs, il choisit celui possédant le poids le plus élevé.

S'il ne trouve aucun candidat, le dominé va devenir actif, comme pendant la phase de construction. Si un dominant tombe, il se peut donc que tous ses dominés entrent en mode actifs. Ils vont donc élire le ou les meilleurs candidats pour devenir dominant, comme à la phase de construction. Le temporisateur τ (section 2.3.2 p34) de l'élection permet d'être certain que le processus converge. Les dominants élus seront probablement déconnectés, ils rentreront donc dans le processus de maintenance pour dominant.

2-Reconnexion des dominants La maintenance pour les dominés n'est pas du tout adaptée aux dominants. En effet, un dominant peut chercher un autre dominant dans son voisinage, mais il n'est pas sûr que ce dernier soit connecté. Il se peut que ce dominant soit un de ses descendants. S'il le fixait comme père, il créerait donc des boucles au sein de son MCDS, phénomène à bannir. Il est donc nécessaire d'établir une reconnexion explicite par l'envoi de messages.

Deux reconnexions sont possibles : soit le père tente de se reconnecter à ses fils, soit les fils tentent de se reconnecter à un nouveau père. La première solution est à écarter, les fils pouvant être mobiles, peut être dans des directions différentes. Chaque dominant doit donc être responsable de trouver un père.

Deux dominants sont éloignés de $2k_{mcds} + 1$ sauts au maximum. Un dominant déconnecté va donc envoyer une demande de reconnexion, un `recon-req` (tab. 5), avec un TTL de $2k_{mcds} + 1$. Il peut exister plusieurs dominants déconnectés dans une zone, espacés de

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Source							
Destination							
Relais							
Type		TTL	Identifiant				
Identifiant AP-Hello							

Table 5: Format du `reconnect-req`

$2k_{mcds} + 1$ sauts. Cependant, il existera obligatoirement en lisière de cette zone, un dominant pouvant se reconnecter. De même, il se peut qu'une branche entière soit cassée, la reconnexion du dominant père entraînera la reconnexion de tous ses descendants.

Seul un dominant connecté doit avoir le droit de répondre à un `recon-rep`, du même format que le `join-message` (tab. 3). Il faut donc que ce dominant possède des informations plus récentes à propos de la racine du MCDS. Le dominant déconnecté envoie donc avec son `recon-req` l'identifiant du dernier `ap-hello` entendu. Seul un dominant ayant reçu un `ap-hello` avec un identifiant plus grand a le droit de répondre. Il répond avec un `recon-rep` dans lequel il donne entre autre son poids et le dernier identifiant entendu. Ce `recon-rep` suit ensuite le même principe qu'un `join-message`, pour l'invitation à la connexion des dominants.

Nous souhaitons éviter une inondation des demandes de reconnexion. Nous avons donc introduit un mécanisme particulier. Un `recon-req` est envoyé en broadcast par un dominant. Ses dominés le relaient en broadcast. Ensuite, les dominés possédant un autre dominant vont relayer en unicast vers leur propre dominant. On évite d'inonder ainsi tout le voisinage, limitant au plus possible l'impact, tout en ne limitant pas pour autant l'efficacité d'une telle reconnexion.

Pour détailler les échanges de paquets, la demande de reconnexion d'un nœud A suit le schéma suivant:

1. A envoie en broadcast un paquet `recon-req`.
   Les dominés de A le relaient en broadcast.
   Les autres dominés, x, le relaient en unicast vers leur dominant (D_x).

2. D_x, s'il est connecté, répond avec un `recon-reply`.
   La réponse suit la route inverse en unicast.

3. A reçoit le paquet de réponse de D_x, refixe son père et renvoie une réponse `recon-advert` afin que tous les intermédiaires deviennent dominants.
   Chaque nœud intermédiaire relaie le paquet et fixe ses nouveaux père et état.

Il est obligatoire que le processus se déroule en 2 phases, la première phase étant nécessaire si plusieurs dominants répondent.

Le `recon-req` doit suivre le même chemin que le `recon-reply` et le `recon-advert`. Il existe deux possibilités pour forcer ces routes. Les intermédiaires peuvent temporairement mémoriser les nœuds précédents et suivants, et l'identifiant du paquet. Les réponses suivront le même chemin. Cette solution permet de minimiser la bande passante nécessaire, mais demande une certaine capacité de stockage. La deuxième solution consiste à accumuler les adresses des intermédiaires directement dans le paquet. Cette solution crée des paquets plus gros mais ne demande aucun stockage temporaire et temporisateur de péremption associés.

3-Cassage du MCDS Si l'environnement est très dynamique, s'il existe de trop nombreuses branches déconnectées, il se peut que le MCDS ait du mal à se reconnecter. Nous avons donc créé un algorithme afin de détruire la branche et la reconstruire de façon plus efficace. Un dominant qui aura tenté n_{max} reconstructions infructueuses, va détruire sa branche : il enverra un `break-message`. Chacun de ses fils ou dominé, recevant un tel message va se réinitialiser et va relayer le paquet. En suivant un ordre descendant, l'intégralité de la branche sera réinitialisée. Un nœud se réinitialise en remettant à zéro toutes ses connaissances sur le MCDS, et en revenant à l'état *idle*. Deux dominants doivent éviter de casser simultanément des branches différentes. En effet, une seule branche cassée peut suffire à par exemple réduire les congestions et donc autoriser les autres branches à se reconnecter. Un temporisateur aléatoire a donc été introduit pour tout dominant souhaitant casser sa branche.

Une zone se retrouve donc dans un état identique à l'initialisation. Cependant, il n'existe dans cette zone aucun leader capable d'initier la reconstruction. Si un nœud intérieur est élu, il faut qu'il se connecte, ce qui n'est pas trivial. Nous avons donc choisi que les dominants connectés jouxtant cette zone jouent le rôle de leader de reconstruction. Si un dominant détecte en cours de fonctionnement qu'il possède un k_{mcds}-voisin d'état *idle*, il va envoyer un `join-message`, comme lors de la construction. Un nœud *idle* recevant un tel message va devenir *dominant* et se raccrocher au dominant l'ayant invité. Ce nouveau dominant va entraîner des changements en cascade, comme lors de la construction.

Il se peut dans le cas pire que la zone des nœuds *idle* soit à exactement $k_{mcds}+1$ sauts d'un dominant. Il est donc nécessaire qu'un dominé, 1-voisin de son dominant, et possédant un voisin *idle* à exactement k_{mcds} sauts, avertisse son dominant qu'un nœud *idle* a besoin de se reconstruire et qu'il faut qu'il envoie un `join-message`.

Suppression des dominants inutiles Si nous ne prévoyons pas un mécanisme de suppression de dominants, la cardinalité du k-MCDS va avoir tendance à augmenter au cours du temps. Nous avons donc choisi de périodiquement vérifier la nécessité d'un dominant. S'il ne possède pas de dominés à k_{mcds} sauts exactement, il deviendra dominé de son ancien père. Il enverra un message de changement d'état, forçant ses dominés présents à moins de k_{mcds} sauts de choisir ce père comme nouveau dominant.

Il est nécessaire cependant qu'un dominant ne se déclare pas inutile trop vite, un dominé n'ayant pas encore eu le temps par exemple de se déclarer auprès de son père. Un dominé se déclarant au bout de maximum max_{retry} paquets `hellos`, un dominant inutile doit attendre

avant de pouvoir devenir dominé un temps $t_{inutile}$:

$$t_{inutile} = max_{retry} \cdot intervall_{hello} \tag{18}$$

2.4 Clusters

2.4.1 Intérêt

AP virtuels Nous souhaitons créer des zones de regroupement de nœuds proches géographiquement, zones que nous appellerons *clusters*. Une telle structure viendra se combiner avec la structure du MCDS. En effet, le MCDS nous sert de backbone pour acheminer tout le trafic de contrôle. Les clusters, eux, peuvent permettre de créer des zones avec un chef de cluster, appelé *clusterhead*, chargé d'assurer un certain nombre de services. Son rôle pourrait être semblable au rôle d'un *AP-virtuel*: prise en charge des enregistrements Mobile IP des mobiles, du paging pour les mobiles en économie d'énergie, de l'attribution d'adresses... Il doit être membre du backbone pour des critères de stabilité identiques au MCDS et de diffusion efficace des informations à ses membres et aux autres clusterheads.

D'autre part, nous souhaitons construire des zones locales dans lesquelles le routage sera proactif. Nous optimisons les communications locales qu'elles soient vers un nœud destinataire appartenant à cette zone, ou que la destination ne soit par exemple qu'un relais vers Internet. Cependant, la taille de ces zones permet de limiter la quantité d'information sur la topologie à échanger et mémoriser. Souvent, un utilisateur communique principalement avec des personnes géographiquement proches (collègues, amis...), sinon, il peut joindre des destinataires lointains par exemple via Internet. Nous estimons que les communications intermédiaires sont peu fréquentes. Nous préférons donc optimiser le routage au sein d'une zone et interconnecter ces zones, quite à ce que les routes inter-zone ne soient pas optimales, le coût de la connaissance de toute la topologie du réseau hybride étant extrêmement important (nous devons connaître l'intégralité de notre $k_{k=\infty}$-voisinage).

Taille Il convient cependant de choisir un diamètre optimal des clusters, que nous noterons $k_{cluster}$. Nous connaissons notre k-voisinage, il est donc intéressant d'utiliser toutes ces informations, d'où $k_{cluster} \geq k = k_{mcds}$.

Un mobile doit se réenregistrer à chaque changement de zone, il faut donc que ces zones soient relativement étendues pour limiter la surcharge réseau de réenregistrement. D'autre part, le mobile doit se réenregistrer auprès d'un nouveau maître chaque fois qu'il sort de sa zone de paging. Nous souhaitons donner au clusterhead le rôle de maître, il faut donc que $k_{cluster}$ soit relativement grand. Nous pensons également donner au clusterhead le rôle d'assignation d'adresses temporaires globales. Une zone posséderait donc un regroupement logique de ses adresses dans un même sous-réseau, de façon semblable au filaire. Pour être intéressant, il faut qu'il regroupe un minimum de nœuds. Enfin, nous souhaitons faire du routage inter-clusters. Aussi, pour que la clusterisation apporte un plus par rapport au routage à plat, il faut un $k_{cluster}$ important.

La construction de clusters présente indubitablement un coût relatif à leur diamètre. Nous devons échanger à la construction des clusters des informations sur le $k_{cluster}$-voisinage, ce qui demande à chaque nœud du backbone d'envoyer un `hello`, pour le propager sur $k_{cluster}$ sauts. De même, à la maintenance, il nous faut vérifier la présence de notre clusterhead, mécanisme d'autant plus coûteux que ce chef est éloigné. Il faut donc trouver le bon compromis de taille, ce paramètre étant à optimiser selon les applications.

Enchaînement des constructions Nous avons choisi de construire les clusters postérieurement au MCDS. En effet, nous pouvons ainsi utiliser notre backbone pour construire les clusters et donc ne faire participer que les nœuds dominants, en optimisant le trafic de broadcast. Tous les nœuds dominés choisissent en outre le clusterhead de leur dominant, ils n'ont ainsi pas besoin de participer à l'élection, ce qui réduit la charge réseau. De plus, comme le clusterhead a pour but d'échanger un trafic de contrôle avec ses membres, il est logique de construire d'abord le backbone puis les clusterhead dessus pour borner la distance maximale à parcourir pour les messages de contrôle empruntant le backbone.

2.4.2 Construction

Nous avons choisi de ne pas utiliser la méthode de construction de [25], s'appuyant directement sur le MCDS. En effet, comme nous l'avons dit la maintenance est complexe et coûteuse. Elle nécessite de plus une mise à jour constante des membres pour que chaque clusterhead sache s'il peut encore accepter de nouveaux membres. Une telle connaissance introduit un surcoût de messages, une convergence plus lente d'autant qu'il est relativement difficile de déterminer qu'un membre est parti, qu'il subit des collisions, que le MCDS est cassé...

Initiation Pour la clusterisation, nous avons besoin d'un coordinateur qui décide d'initier la construction des clusters. Nous avons naturellement choisi l'AP comme leader. La décision est propagée grâce aux paquets `hellos`, dans lesquels un drapeau est fixé à 1 lors de l'initialisation. Un nœud peut donc savoir quand il s'allume s'il doit se lancer en procédure de création (tous ses voisins envoient un drapeau nul), ou au contraire dans une procédure de maintenance (drapeau égal à un).

Il faut initier la construction des clusters après celle du MCDS. Cependant, il nous est impossible de borner ce temps puisque nous ne pouvons connaître à l'avance la taille totale du réseau. Nous ne propageons donc la décision que si notre état est déjà déterminé (dominé ou dominant), ce qui constitue la première étape de la construction du MCDS. Lorsqu'ils reçoivent la décision d'initialisation, il existe de toutes façons un temporisateur permettant aux nœuds nécessaires à la connexion des dominants de la première étape de se déclarer. Un dominé relaie le message de connexion de son dominant puis doit attendre que tous ses voisins actifs devenus dominants aient pu faire part de leur vœu de connexion. Il est donc nécessaire d'attendre τ (cf. section 2.3.2 p34).

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Source							
Destination							
Type			Poids		Identifiant		
TTL							
Relais							
Clusterhead							

Table 6: Format des paquets `cluster-hello`

Découverte de voisinage Nous construisons un $k_{cluster}$-cluster. Cependant, les dominés choisissent automatiquement le clusterhead de leur père. Un dominant devra donc choisir un clusterhead à $k_{cluster} - k_{mcds}$ sauts pour pouvoir servir tous ses dominés. Comme nous l'avons expliqué, la construction de cluster utilise le MCDS. Un nœud est donc voisin d'un autre s'il est à un saut dans l'arbre, le voisinage n'est donc plus physique. Il nous faut construire une autre table de voisinage, utilisée seulement pour la construction des clusters, et donc non maintenue pendant un long moment. Il suffit donc d'envoyer des `hellos` périodiques jusqu'au choix d'un clusterhead pour chaque voisin de cette table. Nous avons choisi d'envoyer un paquet `cluster-hello` (fig. 6) tous les $intervall_{fast-hellos}$, ce qui correspond à la découverte rapide de voisinage.

Élection des clusterheads Cette étape est exécutée après la découverte du voisinage virtuel du MCDS. Pour débuter la construction des MCDS, nous armons donc un temporisateur qui nous permet d'être certains que tous les nœuds du voisinage se sont déclarés. Nous estimons que le voisinage est créé après réception d'un paquet valide de `cluster-hello`, envoyé au maximum max_{retry} fois, les renvois étant espacés de intervall$_{fast-hello}$, d'où :

$$t = max_{retry} \cdot intervall_{fast-hello} \tag{19}$$

En cas d'environnement très perturbé, il peut exister plus de collisions, mais attendre plus de paquets de `cluster-hellos` implique une convergence plus lente tout en augmentant les probabilités globales de collisions de paquets sur le médium. Le nombre de clusterheads sera donc plus important, mais ils pourront remplir leur rôle. Cependant, pour limiter le fractionnement des clusters, nous pouvons attendre que notre père et nos fils soient présents dans notre table de voisinage virtuel avant d'élire les clusterheads.

Nous souhaitons minimiser le nombre de clusterheads. Il ne faut donc pas qu'un nœud décide seul de l'identité de son clusterhead et la propage. D'une part, nous aurons des zones non connexes, puisqu'un intermédiaire peut choisir un autre nœud de poids plus fort que la source ne voit pas (fig. 3 p15). D'autre part, le nombre de clusterheads est souvent non optimal, par exemple dans le cas d'une chaîne à poids croissant (fig. 7).

Un nœud va donc se déclarer clusterhead s'il possède le poids le plus élevé parmi tous ses voisins ne possédant pas de clusterhead. Il se peut donc qu'au cours d'une étape, il

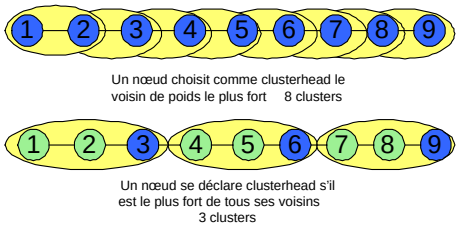

Figure 7: clusterisation d'une chaine de nœuds de poids croissant

n'y ait pas d'élection au sein du voisinage local d'un nœud, chacun attendant la décision d'un nœud de plus fort poids. Cependant, de proche en proche, le processus convergera. A chaque tour, un nœud au moins s'élira leader, et tous les nœuds à moins de $k_{cluster}$-k_{mcds} sauts le choisiront comme clusterhead. Ensuite, le nouveau nœud de poids le plus fort s'élira clusterhead, et ainsi de suite. Cependant, ceci constitue un cas particulier, il est beaucoup plus fréquent que plusieurs nœuds soient élus clusterheads au cours de la même étape lorsqu'il n'y a pas de situation d'interblocage.

Chaque clusterhead envoie au maximum un message de décision de clusterhead à son $k_{cluster} - k_{mcds}$ voisinage, relayé par chacun, ce qui donne une première complexité en messages en $O(D \cdot \Delta_k)$, avec Δ_k le nombre de k-voisins et D le nombre de dominants. D'autre part, chaque nœud envoie périodiquement un `cluster-hello` jusqu'à ce que tous ses voisins aient choisi un clusterhead. Il peut exister des interblocages sur le voisinage du k-MCDS. Si la hauteur[13] du k-MCDS est de h, la chaîne de longueur maximale pouvant créer des interblocages est au maximum de $2 \cdot h$ sauts. De plus, chaque élection prend t temps (eq. 19). S'il existe un interblocage total, tous les nœuds émettent des `cluster-hellos` jusqu'à la fin de la construction. La complexité totale en messages est donc :

$$C_{messages} = O\left(D \cdot \Delta_k + \frac{\frac{2 \cdot h}{k_{cluster} - k_{mcds}} \cdot t}{intervall_{fast-hellos}} \cdot n\right) \tag{20}$$

De même la complexité en temps est, si on néglige les délais de propagation des paquets de décisions:

$$C_{temps} = O\left(\frac{2 \cdot h}{k_{cluster} - k_{mcds}} \cdot t\right) \tag{21}$$

Enfin, comme nous l'avons déjà dit, les dominés choisissent le clusterhead de leur dominant. Les zones de dominance sont obligatoirement connexes (sinon, les dominés ne pourrait pas recevoir par exemple les `bts-hellos` venant de leur dominant), donc la connexité des clusters est également vérifiée pour les dominés.

[13]la hauteur d'un arbre (MCDS ici) est la distance entre la racine et la feuille la plus éloignée

2.4.3 Maintenance

La maintenance des clusters est une procédure obligatoire dans un environnement dynamique tel que les réseaux ad hoc. Cependant, une telle problématique n'est pas du tout abordée dans [4, 15, 25], la complexité de [33] est élevée, et sa convergence n'est pas étudiée. Il est absolument nécessaire que les clusterheads soient le plus stable possible et donc que les procédures de maintenance favorisent les nœuds déjà en place. En effet, ils renferment l'intégralité des informations nécessaires au fonctionnement d'un cluster (adresses de chaque nœud, gestion de la mobilité, informations globales...) Leur remplacement nécessite donc soit un transfert des informations de l'ancien clusterhead vers le nouveau, soit une procédure de réenregistrement des membres du cluster, ce qui peut être problématique en terme de charge, de réveil des nœuds endormis...

Maintien des informations de clusterisation Un nœud a besoin de savoir quelle est l'adresse de leur clusterhead et le nœud servant d'intermédiaire pour le joindre. Il connaîtra à terme l'ensemble des nœuds de son cluster par l'intermédiaire d'un protocole de routage de type OLSR. Cependant, dans l'absolu, il n'a pas besoin de connaître son $k_{cluster}$-voisinage du MCDS. Aussi, un nœud ne peut optimiser son clusterhead grâce aux informations de voisinage, puisque ses informations étant partielles, il peut ne pas savoir qu'un clusterhead est candidat potentiel, c'est pourquoi nous avons développé des procédures de maintenance particulières.

Détection de connexion avec le clusterhead Un nœud doit pouvoir savoir si le clusterhead qu'il a choisi est encore accessible à moins de $k_{cluster} - k_{mcds}$ sauts. Cependant, il n'a aucun moyen de tirer de telles informations de son voisinage. Nous avons donc mis en place un mécanisme semblable à celui utilisé pour la détection de connexion au leader (AP). Chaque clusterhead va émettre périodiquement un `clusterhead-hello`. Si un nœud rate plus de deux de ces messages, il peut considérer qu'il est déconnecté et chercher un autre clusterhead. S'il se trouve qu'il est en fait toujours connecté, il recevra prochainement un nouveau `clusterhead-hello` et pourra stopper sa recherche. Globalement, nous minimisons le temps de reconnexion d'un nœud, sans toutefois créer une instabilité avec des nœuds oscillant entre deux clusterheads. Chaque nœud va relayer le `clusterhead-hello`, s'il ne l'a pas déjà vu, en marquant son adresse dans le champs source. Ceci permettra à un nœud quelconque de connaître l'intermédiaire lui permettant de contacter son clusterhead.

Reconnexion Un nœud peut demander une reconnexion explicite en envoyant un `reconnect-request`. Ce paquet contient l'identifiant du clusterhead actuel et l'identifiant du dernier `clusterhead-hello` entendu. Cependant, il est important que les clusters obtenus, et ce même après maintenance, soient connexes. Donc un nœud qui possède le même clusterhead peut relayer le paquet sans restriction. Un tel paquet est relayé en broadcast sur le MCDS pour maximiser les chances de réception par le clusterhead : la direction dans laquelle devrait se situer le clusterhead n'est pas stipulée. Un nœud peut également relayer

4bits	4bits	4bits	4bits	4bits	4bits	4bits	4bits
Clusterhead							
Source							
Destination							
TTL							
Type			Poids		Identifiant		

Table 7: Format des paquets `clusterhead-hellos`

le paquet s'il appartient à un clusterhead différent mais jouxte le demandeur. Dans ce cas là, l'hôte relaie le paquet vers son propre clusterhead en unicast. Donc, si un nœud possède un clusterhead différent de celui du paquet mais que la destination est en broadcast, il ne relaie pas ce paquet.

Un nœud D peut répondre à la source S de la requête :

- S'il possède le même clusterhead C, que C est encore valide, et que le dernier `clusterhead-hello` réceptionné possédait un identifiant supérieur à celui inscrit dans le `reconnect-request` de S ;
- S'il possède un clusterhead C différent de celui de la source S, que C est encore valide (entendu le dernier `clusterhead-hello`) et qu'il est assez proche de S : $dist_{S \to D} + dist_{D \to C} < k_{cluster} - k_{mcds}$.

Lorsque le dominant déconnecté reçoit une réponse, il arme un temporisateur, lui permettant de stocker plusieurs réponses si d'autres arrivent. Lorsque le temps est écoulé, il choisit la réponse venant du clusterhead à plus fort poids. Il optimise donc la stabilité de son clusterhead.

Optimisations

Suppression des clusterheads inutiles Il se peut que certains clusterheads soient inutiles, c'est à dire qu'ils ne desservent aucun nœud et qu'ils puissent de plus se connecter à un autre clusterhead. Pour une telle optimisation, nous pouvons utiliser la propriété de connexité des clusters (section 2.4.2). Si aucun des voisins de C n'a choisi C comme clusterhead, alors a fortiori, aucun nœud, éloigné d'un nombre de sauts quelconque, n'a choisi C comme clusterhead. Donc, un clusterhead rentre en état *inutile* si aucun de ses voisins ne l'a choisi, et il se rattachera au propriétaire du prochain `clusterhead-hello` qu'il recevra. Il tire des paquets `hellos` les clusterheads de ses voisins.

Choix d'un clusterhead de poids plus fort à la réception d'un clusterhead-hello Un nœud à la frontière de plusieurs clusters va recevoir un `clusterhead-hello` de chaque zone. Si les clusterheads entendus sont à moins de $k_{cluster} - k_{mcds}$ sauts, il a donc

la possibilité de choisir le meilleur, le meilleur étant celui de poids le plus élevé. Ainsi, un clusterhead de poids élevé aura tendance à posséder plus de membres, et de proche en proche les clusterheads de poids faible pourront devenir inutiles et être remplacés. De plus, le choix d'un clusterhead de poids élevé favorise la stabilité de la structure, notre poids changeant peu au cours du temps.

Pour éviter un changement trop fréquent de clusterhead, un nœud n'engage le changement que si la différence de poids avec son ancien clusterhead est significative. Nous avons fixé arbitrairement cette différence à 30%, mais elle dépend bien sûr de la métrique utilisée.

3 Résultats

3.1 Modélisation

Simulateur Nous avons choisi de développer notre solution sous Opnet Modeler 8.1[14], en utilisant au niveau 2 la couche 802.11b intégrée au simulateur. Opnet Modeler permet de spécifier les formats de paquets, de collecter des statistiques et fonctionne sous un mode événementiel.

Chaque nœud est représenté indépendamment, le simulateur estimant si un lien radio existe entre deux nœuds quelconques en fonction de la distance qui les sépare, de la puissance d'émission... Un nœud est donc modélisé comme un ensemble récepteur/émetteur, une couche 802.11 reliée au dispositif d'émission/réception, et une couche MCDS-Cluster au dessus de la couche 802.11b (figure 8). Un processus à part permet de simuler la mobilité d'un nœud, et un processus permet de jouer l'interface entre la couche MAC et notre couche.

Chaque processus est représenté comme un diagramme d'état: lorsqu'on rentre dans un état, une série d'instructions propres est exécutée, de même que lorsqu'on sort d'un état. Lorsqu'un évènement survient (interruption, arrivée d'un paquet...), on va sortir de l'état et suivre la transition vraie à ce moment là pour entrer dans un autre état.

Paramètres Un nœud possède une portée radio de 300m, paramètre par défaut du modèle 802.11b de Opnet, et se déplace sur une surface rectangulaire de taille variable, afin de faire varier la densité et le nombre de nœuds intervenant dans la simulation. La vitesse est également un paramètre variable, entre 0 et 25 $m.s^{-1}$. Nous avons simulé les performances du réseau ad hoc pour un nombre de nœuds allant de 5 à 50, leur position initiale étant aléatoire sur la grille. Un nœud possède une adresse MAC et une adresse ad hoc. Nous avons choisi de fusionner les deux dans notre simulation, puisqu'elles ne présentent pas de différence pour notre étude.

Nous collectons différentes statistiques pour l'évaluation de notre solution. Pour étudier le MCDS, nous mesurons sa cardinalité au cours du temps, l'overhead introduit, le temps de connexion, le nombre de tentatives de reconnexion et le nombre de paquets induits. Nous collectons les mêmes statistiques pour les clusters. Enfin, nous considérons qu'un nœud

[14] cf. *http://www.opnet.com/products/modeler/*

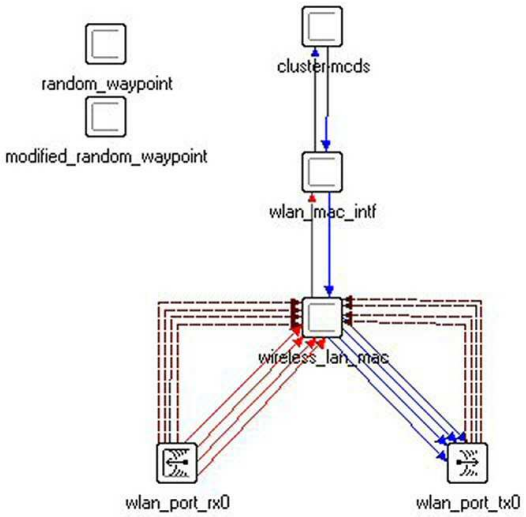

Figure 8: Modélisation des couches de transmission

est déconnecté d'un clusterhead/dominant s'il rate plus de deux *hellos* consécutifs, ce qui surestime donc les temps de déconnexion (des paquets non acquittés peuvent se perdre pour des raisons de collisions).

Mobilité Nous avons choisi pour simuler le mouvement de nos nœuds le *random waypoint model*, décrit dans [12]. Un nœud choisi au temps 0 une destination d et une vitesse v aléatoire. Une fois arrivé au point fixé en se déplaçant à la vitesse tirée, il va choisir un autre couple aléatoire vitesse/destination. Nous avons par contre effectué quelques modifications. Tout d'abord, un nœud pouvant tirer une vitesse nulle, il ne parviendra jamais à la destination choisie. Le nœud ne restera donc avec une vitesse nulle que pendant t temps au maximum. D'autre part, nous préférons modifier le type de tirage de la vitesse. le nœud, initialement, tire bien une vitesse aléatoire. Cependant, quand il doit en retirer une nouvelle, il va utiliser une loi normale centrée sur son ancienne vitesse, d'écart la vitesse maximale sur 10. Nous trouvons cette vitesse variant moins brutalement plus réaliste.

3.2 Résultats

Nous avons mesuré l'influence de nombreux paramètres sur les propriétés de notre structure. Un seul paramètre est libre au cours d'une série de simulation. Les autres sont fixés à des valeurs que nous considérons comme usuelles :

- Vitesse : $5m \cdot s^{-1}$;

- Nombre de nœuds : 30 ;
- densité : 10 voisins en moyenne ;
- k_{mcds} : 2 ;
- $k_{cluster}$: 4.

3.2.1 Durée

Nous avons choisi dans un premier temps de mesurer le comportement de nos structures sur une longue période de temps. Nous avons donc simulé avec nos paramètres standards les caractéristiques de nos structure pendant une heure de la durée de vie réelle d'un réseau hybride. Certains résultats, non graphiques, ne sont donnés que dans les commentaires.

En moyenne, nous avons en moyenne 7,7 dominants au cours de l'heure de simulation, ce qui représente le quart des nœuds. De plus, environ la moitié de ces dominants ne possèdent aucun dominés, il ne servent qu'à l'interconnexion du backbone, dépensant donc peut être à terme moins d'énergie puisqu'ils ne délivreront pas de réels services, en dialoguant activement avec leurs dominés. Cette cardinalité du k-MCDS nous semble donc acceptable. Un dominant garde son rôle de dominant durant environ 2,5 minutes. De même, un dominé garde un même dominant durant environ 4,3 minutes. Notre k-MCDS présente donc des propriétés de stabilité intéressantes. Ces durées de persistance sont relativement élevées pour que le dominant puisse assurer son rôle pendant un laps de temps suffisant, sans multiplier les réaffiliations. Cependant, ce temps ne nous semble pas excessif, afin que les dominants n'épuisent pas non plus leurs réserves en énergie. Nous pouvons observer la cardinalité de notre k-MCDS au cours du temps (fig. 9). Les brusques cassures correspondent à des cassures du k-MCDS, diminuant la cardinalité brutalement. Cependant, les variations au cours du temps nous semblent rester acceptables, le nombre moyen de dominants variant dans des proportions raisonnables. On retrouve le même comportement sur l'estimation de la connexité au k-MCDS (fig. 10). Les faibles taux de connexion correspondent au cassures de notre backbone, un petit temps étant nécessaire pour reconstruire la zone. Le taux de connexion global au k-MCDS représente 92,7%, ce taux étant sous-estimé. Cependant, à terme, un nœud vérifiera sa connexité avec les `ap-hellos` et `hellos`, mais à terme, il pourra utiliser tous les paquets de contrôle et de données, ce qui augmentera la quantité d'informations disponible et donc la conexité globale. Enfin, nous pouvons nous apercevoir que les dominants élus sont les nœuds ayant tendance à avoir le plus fort poids (tab. 9). De même, les dominants à plus fort poids prennent en charge les dominés, alors que les autres ne servent qu'à l'interconnexion.

Pour la clusterisation, nous avons en moyenne 3.1 clusterheads, ce qui représente le dixième des nœuds. Il existe donc peu de chefs dans notre réseau. Un clusterhead garde son rôle durant environ 1,5 minutes. Un dominant, garde lui le même clusterhead pendant environ 4,1 minutes. Les clusterheads changeant sont donc en priorité ceux ne possédant pas de *clients*. Cette stabilité nous parait acceptable. Nous pouvons observer que le nombre de

	MCDS	**Cluster**
Nombre de reconnexions	274	270
Nombre de requêtes avant reconnexion	1.7	2.6
Nombre de paquets par requête	29	4.7
Nombre de cassures	54	
Nombre moyen de nœuds impactés par une cassure	3	

Table 8: Reconnexions et Maintenance du k_{mcds}-MCDS et des $k_{cluster}$-Clusters

Type de nœud	**Poids moyen associé**
Moyenne	56.7
Dominant	70.6
Dominant avec dominé	114.8

Table 9: Poids des différents types de nœuds pour une vitesse de 5m.s^{-1}, $k_{cluster}$=4 et k_{mcds}=2

clusterheads est stable au cours du temps (fig. 9). Enfin, le taux de connexion au clusterhead est plus élevé que pour le k-MCDS, ce qui est logique, un dominant étant en moyenne plus proche de son clusterhead que de l'AP. Un dominant reste en moyenne connecté à son clusterhead durant 93,9%. Cependant, comme pour le MCDS, il serait utile d'optimiser ce taux de connexion en tirant parti des informations qui seront contenues plus tard dans les paquets de données ou contrôle. Le taux de connexion aux clusterheads varie bien évidemment avec les reconstructions du k-MCDS, chaque nœud à ce moment là se réinitialisant (fig. 10). Au vu de ces résultats stationnaires, nous avons effectués la suite de nos simulations sur une durée de 10 minutes.

3.2.2 Mobilité

Nous avons ensuite étudié l'influence de la mobilité sur les performances de nos structures virtuelles. Nous pouvons observer que le nombre de dominants dans notre réseau varie en fonction de la vitesse mais dans des proportions très raisonnables (fig. 11). Elle a tendance au début pour une mobilité croissante d'augmenter le nombre de dominants : les clients changeant souvent, il est nécessaire d'augmenter le nombre de dominants pour maximiser la connexité. En effet, un dominé sinon, aura tendance à avoir plus de difficultés à trouver un dominant à proximité. Cependant, pour de fortes mobilités, supérieures à $15m \cdot s^{-1}$, le k_{mcds}-MCDS a plus de mal à s'adapter, et les performances chutent. Quelle que soit la vitesse, il n'existe en moyenne pas plus de 11 dominants, c'est à dire le tiers des nœuds. Nous pouvons faire exactement les mêmes constatations quant au nombre de clusterheads présents dans le réseau hybride, les deux comportements étant liés. Il existe un maximum de 4 clusterheads pour une vitesse de 5 à 10 $m \cdot s^{-1}$, ce qui représente un 7^{ieme} des nœuds.

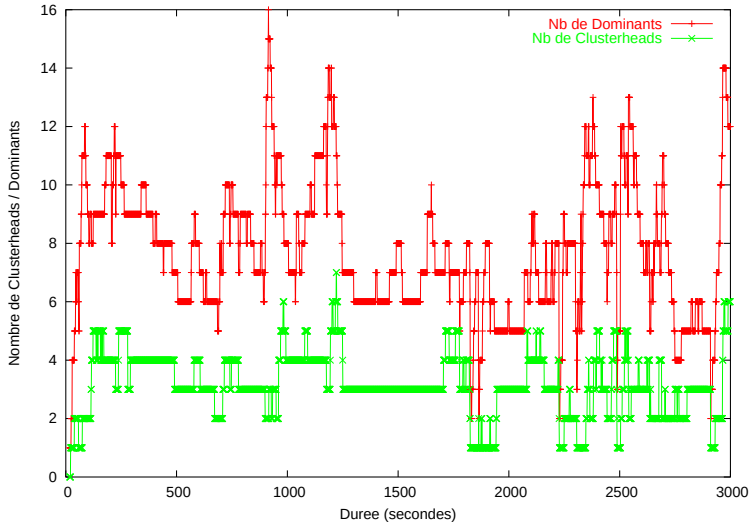

Figure 9: Évolution de la cardinalité de notre structure au cours du temps pour une vitesse de 5m.s^{-1}, k$_{cluster}$=4 et k$_{mcds}$=2

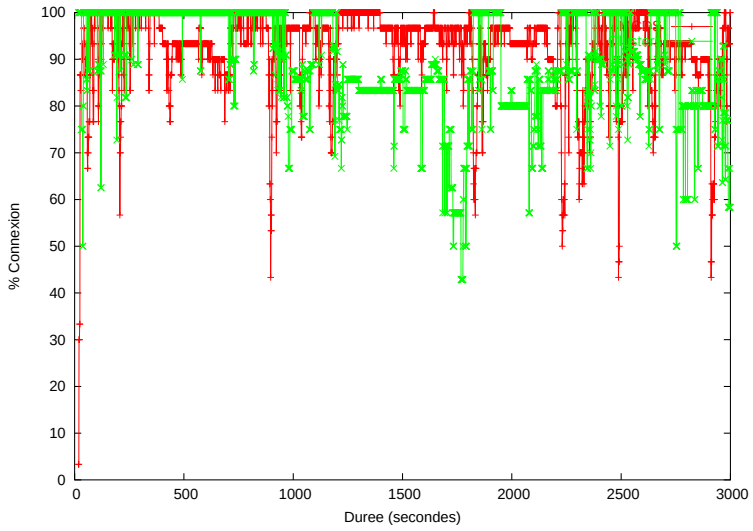

Figure 10: Evolution de la connectivité de notre structure au cours du temps pour une vitesse de 5m.s^{-1}, k$_{cluster}$=4 et k$_{mcds}$=2

La connexité au k_{mcds} reste elle aussi élevée, mais chute brutalement pour des vitesses très élevées, au dessus de 15 ou 20 $m \cdot s^{-1}$. En effet, la connexité au k_{mcds}-MCDS est estimée grâce aux `ap-hellos`. Lors de fortes mobilités, il suffit que les `ap-hellos` soient perdus en début d'arbre pour multiplier les tentatives de reconnexion et sous-estimer le taux de connexion. Cependant, comme nous l'avons dit, nous souhaitons à terme utiliser d'autres types de paquets pour apprécier notre connexion, au moins vis à vis de note père. En effet, les `ap-hellos` resteront nécessaires pour les reconnexions, et éviter à un dominant de par exemple se raccrocher à un de ses descendants. Le taux de connexion devrait donc être à terme plus élevé. En tous les cas, pour des vitesses usuelles inférieures à 7 $m \cdot s^{-1}$, le taux de connexion est supérieur à 90%. Le taux de connexité au clusterhead présente lui de meilleures performances. Le clusterhead étant plus proche, les informations ont moins de chance de se perdre, et la connexion est un peu mieux optimisée. Quelle que soit la vitesse, le taux de connexion reste quasiment toujours au dessus de 95%.

Enfin, la persistance reste appréciable (fig. 13). Même avec de très fortes mobilités, toutes les persistances sont approximativement égales à un, sans chuter brutalement pour des vitesses très élevées. On peut remarquer que pour des vitesses usuelles inférieures à 5 $m \cdot s^{-1}$, un clusterhead reste clusterhead durant plus de 1.7 minute, et ce qui est plus important qu'un nœud garde le même clusterhead durant plus de 2.7 minutes. Il est très importante qu'un client ne change pas de clusterhead trop souvent pour éviter les réaffiliations, la mise à jour des localisation... D'autre part, un dominant reste dominant durant 4 minutes et un dominé garde le même dominant durant lui aussi 4 minutes. Ces persistances pour le k_{mcds}-MCDS nous semblent donc acceptables.

3.2.3 Paramétrage

Nous avons simulé le comportement de nos structures lors de l'évolution du diamètre k_{mcds} du MCDS, distance maximale entre un dominé et son dominant. Le $k_{cluster}$ n'a pas d'impact sur cette série de simulations, puisque le clustering se déroule au dessus de la couche du MCDS, sans l'impacter. Nous remarquons que le nombre de dominants en dehors de la période d'initialisation diminue lorsque k_{mcds} augmente (fig. 14). Lorsque k_{mcds}=4, nous approchons quasiment le diamètre du réseau et seuls 4 dominants suffisent pour former le backbone. Les autres dominants élus se déclarent inutiles au bout d'un certain temps de convergence. Une grande partie des nœuds peut économiser son énergie délégant ses obligations à son dominant. De plus, le paging à terme permettrait d'optimiser la consommation en énergie des nœuds à plus faibles réserves. Le taux de connexion reste toujours supérieur à 92%, mais il a tendance à chuter lorsque k_{mcds} augmente, du fait de l'augmentation de la distance entre un dominé et un dominant. Le nombre de dominants dans notre réseau est donc paramétrable. Dans un réseau fortement mobile, il est préférable de diminuer k_{mcds} pour maximiser les taux de connexion. Pour des vitesses plus élevées, le nombre de dominants peut au contraire être réduit.

Nous avons ensuite simulé l'influence du rayon de nos clusters. k_{mcds} possède une influence sur la clusterisation, nous l'avons donc fixé à 2, ce qui représente selon les simulations précédentes une dizaine de dominants. Les clusters présentent sensiblement le même com-

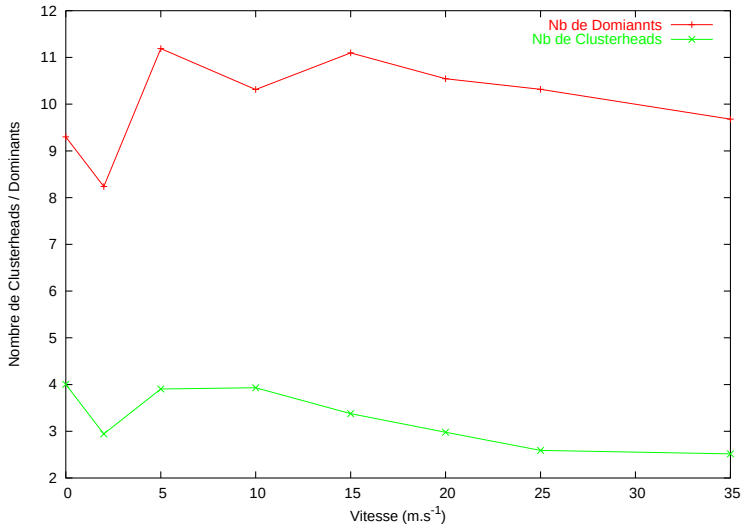

Figure 11: Cardinalité de notre structure en fonction de la mobilité

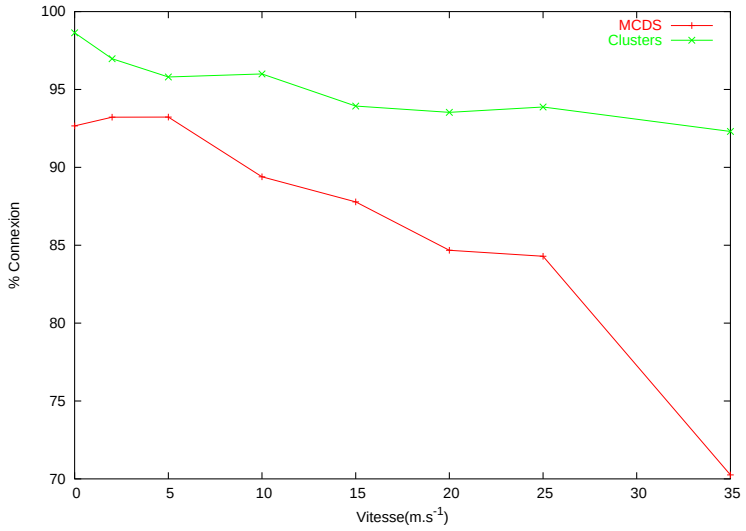

Figure 12: Connectivité de notre structure en fonction de la mobilité

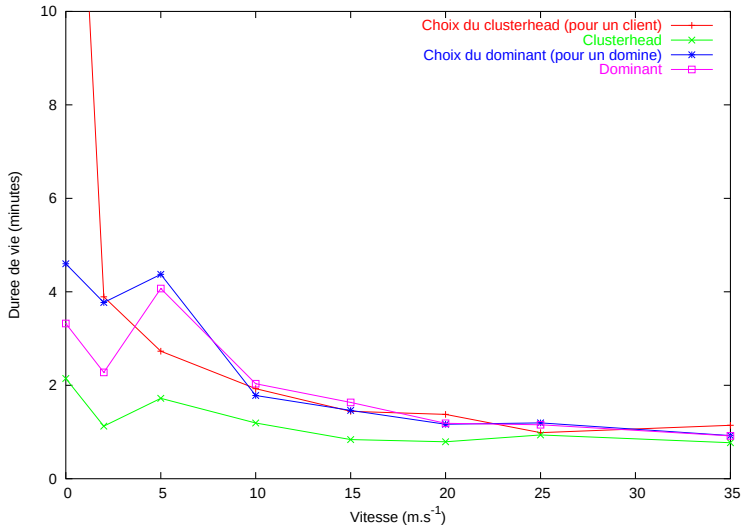

Figure 13: Persistance de notre structure en fonction de la mobilité

portement que pour le MCDS (fig. 15). Lorsque $k_{cluster}$ augmente, le nombre de clusterheads diminue. Lorsque $k_{cluster}$ est fixé à 5, on atteint la taille du 2-MCDS, ce qui explique qu'un seul clusterhead suffise pour tout le réseau. Nous pouvons également remarquer la grande stabilité des clusterheads, leur nombre variant peu au cours du temps. Cette stabilité présente un intérêt, par exemple pour la gestion de la localisation d'un mobile. Le paramétrage possible du nombre de clusterheads est également intéressant, pour adapter la clusterisation à l'application. On peut vouloir limiter le nombre de clusterheads afin de minimiser les réaffiliations, ou au contraire en posséder u pour localiser plus finement un mobile dans le réseau.

3.2.4 Nombre de noeuds

Nous avons ensuite étudié l'influence du nombre de participants actifs sur les performances de notre solution. Le pourcentage de nœuds dédiés au k-MCDS a tendance à augmenter avec le nombre de participants dans le réseau (fig. 16). En effet, les `ap-hellos` doivent subir plus de collisions lorsqu'ils sont éloignés de l'AP, ce qui engendre un surplus de reconnexions et donc de dominants. De plus, lorsque le nombre de participants est très faible, il existe souvent un seul dominant, supprimant ainsi les dominants *d'interconnexion* servant à interconnecter le backbone mais ne desservant aucun dominé. La part de dominants se stabilise au dessous de 35%, en sachant que de nombreux dominants ne possèdent dans ce cas là aucun dominé. La part de clusterheads reste elle quasiment constante, entre 10 et 20%, ce qui nous parait une propriété intéressante.

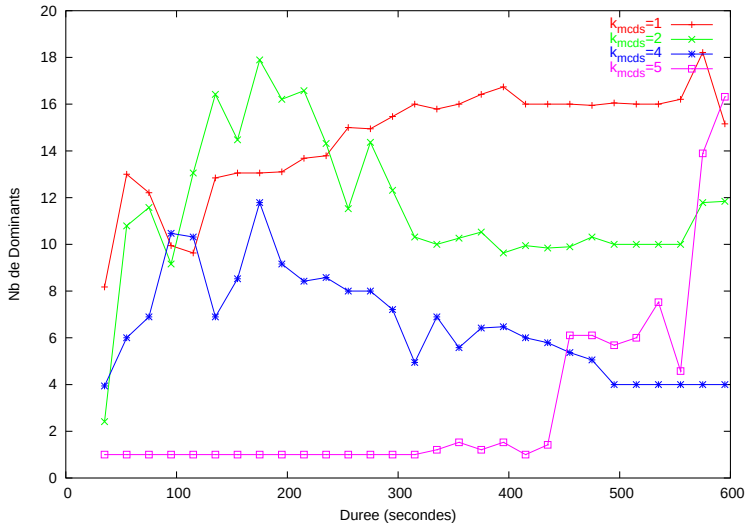

Figure 14: Evolution du nombre de dominants en fonction de k_{mcds}

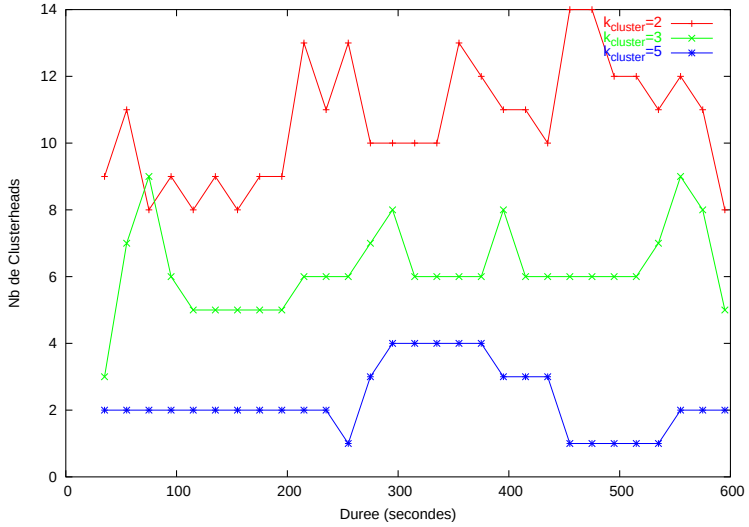

Figure 15: Evolution du nombre de clusterheads en fonction de $k_{cluster}$

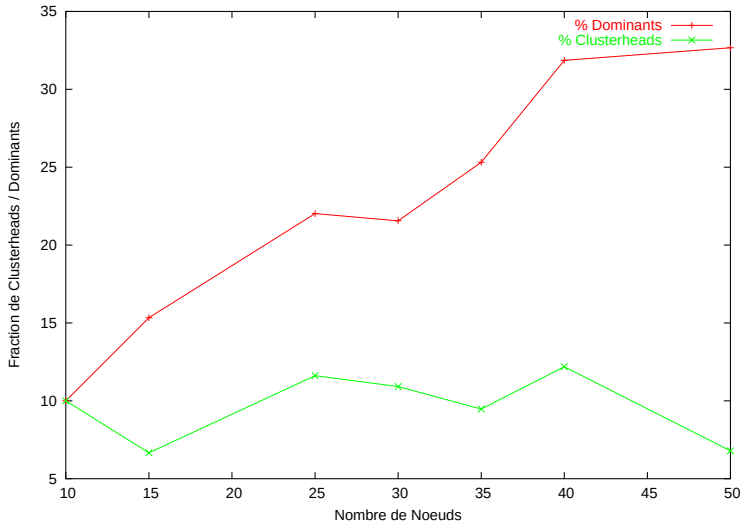

Figure 16: Cardinalité de notre structure en fonction du nombre total de nœuds

La connexité au MCDS et Clusterheads reste relativement stable (fig. 17). Le taux de connexion au MCDS chute lorsque le nombre de participants est élevé du fait de l'allongement moyen du parcours d'un `ap-hello`. Cependant, comme nous l'avons déjà dit, notre solution utilisera à terme les paquets de données pour maximiser cette connexité. Par contre, le taux de connexion au clusterhead reste quasiment constant, au dessus de 93%, ce qui nous parait un taux relativement intéressant.

Enfin, les clusterheads et dominants présentent des persistances très élevées lorsque le nombre de participants est réduit (fig. 18), du fait de la proximité de l'AP et du nombre réduit de collisions. Lorsque le nombre de participants augmente, les persistances diminuent, avec par exemple un nœud qui change de clusterhead toutes les 1.4 minutes dans un réseau de 50 nœuds. Cependant, nous pensons que ces durées de vie restent assez élevées pour être exploitables par certaines applications.

3.2.5 Densité

Enfin, nous avons simulé l'impact du degré moyen, c'est à dire le nombre de voisins. Bien sûr, le nombre de dominants élus a tendance à diminuer quand le degré augmente, puisqu'un nœud ayant en moyenne plus de voisins, un dominant peut desservir plus de dominés (fig. 19). Pour des densités très élevées, il n'existe plus qu'un seul nœud, l'AP, desservant tous les nœuds du réseau. Le nombre de clusterheads suit également la même tendance.

Le taux de connexion au k-MCDS chute pour des densités faibles, au dessous de 7 voisins (fig. 20). En effet, à un tel degré, le nombre de chemins se réduit, et le réseau même peut se

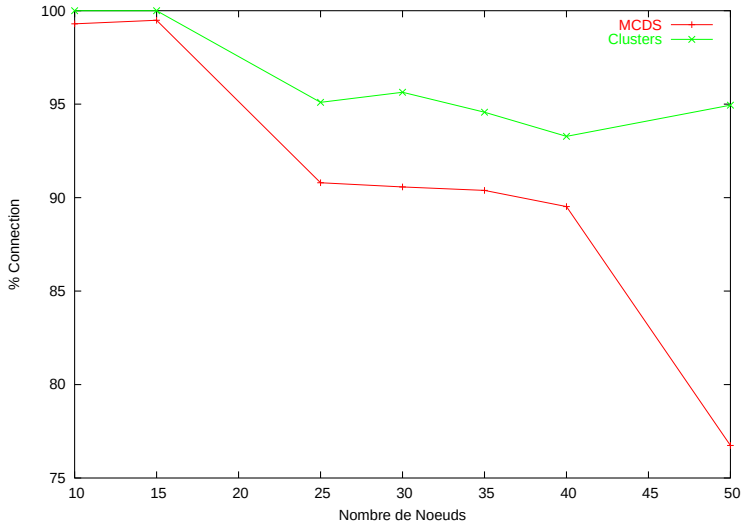

Figure 17: Connectivité de notre structure en fonction du nombre total de nœuds

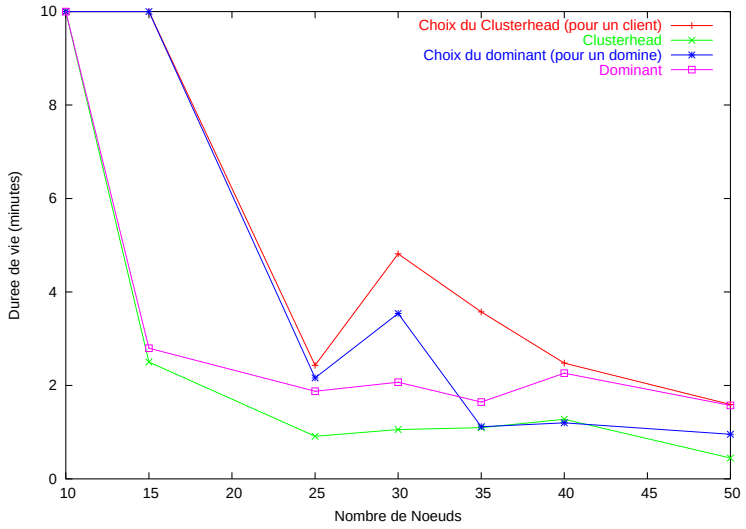

Figure 18: Persistance de notre structure en fonction du nombre total de nœuds

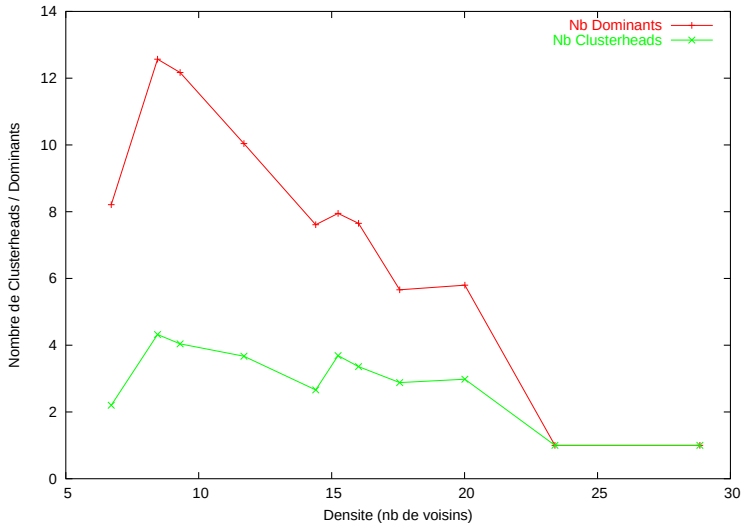

Figure 19: Cardinalité de notre structure en fonction de la densité

partitionner suite au déplacement d'un nœud. La reconnexion est donc plus difficile, et les taux de connexion chutent. De plus, la déconnexion d'un dominant entraîne la déconnexion de toute sa branche, augmentant ainsi l'effet d'une seule deconnexion. La connexité au clusterhead reste elle plus élevée, du fait de l'absence d'un tel phénomène et de la proximité du clusterhead.

Enfin, nous pouvons observer des persistances élevées, quel que soit le degré. Pour un degré très élevé, les persistances très élevées sont dues au nombre réduit de dominants/clusterheads. Pour une densité de 25 voisins, il existe en effet un seul dominant : l'AP. La persistance est donc maximale.

4 Perspectives et Conclusion

Nous avons proposé une méthode de création de backbone et de clusterisation de nœuds d'un réseau hybride afin d'offrir de nouvelles fonctionnalités de localisation, des possibilités de paging...

4.1 Gestion de la mobilité

Nous avons créé une telle structure pour faciliter la gestion de la mobilité au sein de notre réseau hybride. En effet, les paquets `ap-hellos` peuvent aisément fusionner avec les paquets `advertisements` de Mobile IP. Un seul paquet nous permettrait donc d'une part de diffuser

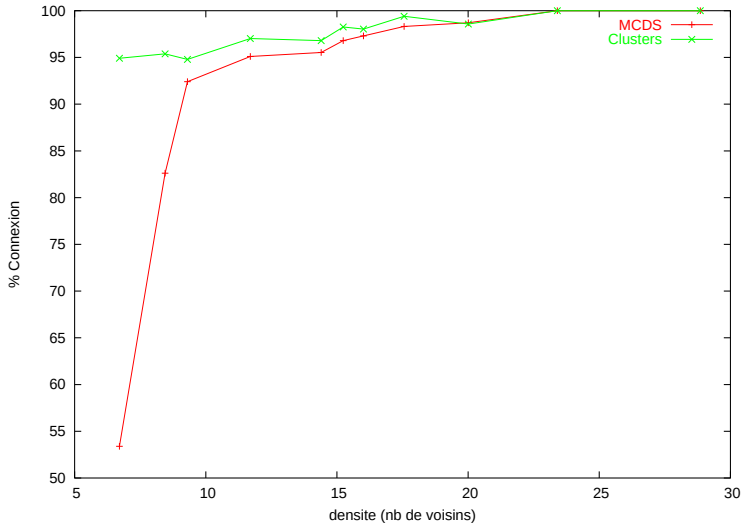

Figure 20: Connectivité de notre structure en fonction de la densité

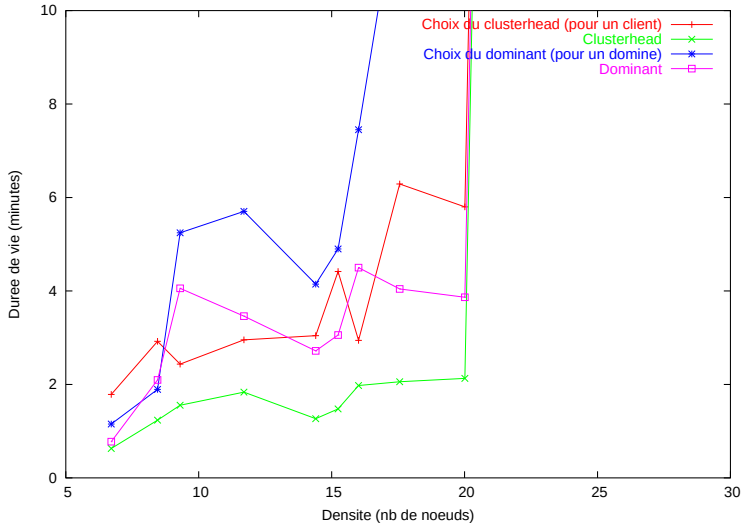

Figure 21: Persistance de notre structure en fonction de la densité

les informations de gestion de la mobilité propres aux paramètres des FA, mais également de maintenir l'arbre. L'overhead de maintien de l'arbre serait donc de beaucoup réduit puisque nous fusionnons des fonctionnalités existantes. D'autre part, les clusterheads ont été introduits afin de gérer la mobilité de leurs membres. Ils correspondraient à de multiples *Foreign Agents* virtuels. Ils enregistreraient l'adresse temporaire de leurs membres et relaieraient leurs paquets vers les routeurs filaires, via l'AP. Nous pourrions de plus utiliser l'arbre MCDS comme dans Cellular IP. Les paquets peuvent être remontés à l'AP en étant passés à l'unique père. Dans l'autre sens, chaque nœud participant au $\text{CIP}_{mcds-cluster}$ maintient des pointeurs entre les nœuds enregistrés et le fils-relais correspondant. Nous souhaitons étudier s'il vaut mieux que seuls les clusterheads participent pour optimiser les ressources en énergie des nœuds faibles du MCDS, ou s'il vaut mieux au contraire faire participer tout le MCDS et optimiser le nombre de sauts. Il faudrait en effet, dans le dernier cas, avant de donner le paquet à l'AP, l'envoyer au clusterhead, peut être plus bas dans l'arbre. Nous souhaitons également implémenter des zones de paging correspondant à la zone de service d'un ou plusieurs clusterheads. Il est dans ce cas nécessaire d'estimer le gain en énergie pour les mobiles endormis, l'overhead des communications entre clusterheads, du membre à son clusterhead, la charge supportable par un clusterhead... Enfin, il serait intéressant d'étudier l'implémentabilité des réseaux mobiles ([23]) dans les réseaux hybrides. L'*Access Router* pourrait correspondre soit à l'AP, soit aux clusterheads, soit aux deux. Les réseaux mobiles, dans ce dernier cas, s'imbriqueraient hiérarchiquement, l'un caché derrière l'autre.

4.2 AP multiples

Un réseau hybride doit pouvoir supporter plusieurs points d'accès pour introduire une tolérance aux pannes, une répartition de la charge, une continuité de services sur une large zone géographique. Nous souhaitons donc créer un MCDS par AP, chaque AP correspondant à la racine. La construction initiale est identique, chaque nœud ne possédant cependant le droit que de participer à la construction d'un seul MCDS. Nous proposons par exemple utiliser l'identifiant de l'AP inscrit dans le `ap-hello` pour déterminer à quel MCDS un nœud appartient. Une fois que les MCDS ont été construits, nous souhaitons créer un recouvrement des arbres. En effet, un nœud oscillant dans les zones frontières changerait constamment d'AP, obligeant à un ré-enregistrement entier. De plus, le routage entre zones proches passerait obligatoirement par les AP puisque de telles zones n'auraient pas connaissance l'une de l'autre. Nous souhaitons donc que les zones frontières diffusent plusieurs `ap-hellos`, dans une zone restreinte. Un nœud sait qu'il est à la frontière puisqu'il entend des nœuds retransmettant des `ap-hellos` d'identifiants différents. Il les diffuse donc avec un TTL restreint. Cependant, pour que la structure soit efficace, il est nécessaire que les MCDS puissent trouver des points de connexion entre eux. Nous souhaitons donc qu'un dominant entendant un autre identifiant de MCDS (directement ou reporté par un de ses dominés), initie une demande de connexion, à la manière de nos *join-messages*. Il obligera donc certains de ses dominés à devenir dominant pour l'interconnexion. Enfin, les deux dominants interconnectés diffuseront dans leur MCDS un avis d'interconnexion pour empêcher leurs voisins d'initier une nouvelle interconnexion au même MCDS. Nous devons cependant

étudier dans le futur l'influence des boucles créées dans les performances du MCDS. La duplication de paquets peut être évitée en introduisant un couple adresse/identifiant unique, similaire à DSR par exemple.

4.3 Zone locale de routage

Nous souhaitons que le routage au sein d'un cluster soit proactif. Chaque nœud d'un cluster connaît ainsi une route optimale vers son clusterhead, son dominant et vers les nœuds proches géographiquement, donc ceux avec lesquels nous pensons qu'il a le plus de chances de communiquer (collaborateurs, voisins...). Cependant, il serait intéressant d'implanter sur notre structure les différents protocoles de routage proactifs existants, d'étudier leur comportement et les optimisations possibles à apporter. En effet, il s'agirait d'optimiser conjointement notre protocole et un protocole de routage, le deuxième pouvant par exemple aider à maintenir le MCDS. Nous pourrions même en fonction des caractéristiques d'un cluster choisir le protocole de routage optimal : une zone dense utilisera plutôt OLSR, une zone moins dense pourra utiliser DSDV... Un cluster choisirait donc à un instant donné son protocole de routage local, indépendamment des autres clusters, accroissant la flexibilité.

4.4 Routage global

Le routage global doit utiliser pour le trafic de broadcast le MCDS afin d'optimiser les retransmissions et diminuer les probabilités de collisions. Les paquets non locaux ne doivent pas obligatoirement passer par les AP. Deux nœuds voisins géographiquement devraient, sinon, passer par les AP pour atteindre leur destination. Les paquets de données ne doivent pas non plus uniquement être acheminés par le MCDS, sans quoi nous créerions des points de congestion.

Nous avons donc choisi d'implémenter dans le futur une solution de routage utilisant le MCDS pour diffuser les paquets de topologie, et les clusters pour l'adressage. Un nœud à la frontière de deux clusters est *nœud passerelle*, c'est à dire qu'il entend des *clusterhead-hellos* d'autres clusters. Les passerelles diffusent dans leur cluster les zones desquelles elles sont adjacentes. Une telle diffusion peut être intégrée aux paquets de mise à jour de topologie du protocole de routage proactif (par exemple les paquets *TC* de OLSR). Le routage s'effectuera donc commme suit :

- Un nœud S demande à son clusterhead C_S la route vers D.
- Si C_S possède déjà la route, il la retourne. Sinon, il diffuse une requête de route dans le MCDS. Lorsque le clusterhead de D, C_D, reçoit un tel paquet, il renvoie une réponse, les adresses des clusters traversés s'ajoutant au paquet.
- S envoie ses paquets au nœud passerelle optimal (délai, proximité, bande passante...) qui jouxte le premier cluster à traverser.
- Chaque passerelle relaie le paquet à la passerelle du cluster suivant.

Nous devons cependant discuter du type de routage proactif/réactif le plus adapté et des paramètres influençant les performances (nombre de nœuds, trafic orienté extérieur, nombre de passerelles,...). Nous devons également arriver à dimensionner le diamètre de nos zones. En effet, les routes suivant approximativement le MCDS, il est nécessaire que les clusters soient de taille relativement importante.

4.5 Conclusion

Nous considérons que notre travail représente un socle modulaire extrêmement flexible pour optimiser les communications de broadcast et gérer la mobilité au sein d'un réseau hybride. Nous avons pris soin d'intégrer de multiples fonctionnalités au sein de notre solution afin de mutualiser les coûts. Nous avons donc proposé une méthode de création d'un MCDS permettant de diminuer le coût du broadcast et d'augmenter sa fiabilité. De plus, ce MCDS nous est également utile à l'intégration de Mobile IP à notre réseau hybride afin de proposer une connectivité Internet à tout hôte le souhaitant. Nous avons créé des clusters afin d'agréger des nœuds proches géographiquement et de proposer des fonctionnalités de paging, d'attribution d'adresses grâce aux clusterheads. Nous avons également présenté une méthode de maintenance pour chacune de ces structure, nécessaire dans un environnement dynamique. Nos MCDS et Clusters présentent une stabilité importante, même avec un nombre significatif de nœuds et une mobilité élevée. La métrique que nous avons introduit semble donc bien se comporter. De plus, la taille de notre backbone et de nos clusters est flexible grâce aux paramètres k_{mcds} et $k_{cluster}$, ce qui rend une adaptation à l'environnement possible. A terme, les clusterheads dialogueront avec la ou les AP pour déterminer les paramètres utiles du réseau hybride et se chargeront de les diffuser à leurs membres. Nous reproduirons donc l'organisation hiérarchique des réseaux filaires, permettant ainsi l'adaptation d'un grand nombre d'applications existantes.

References

[1] Sungjoon Ahn and A. Udaya Snakar. Adapting to route-demand and mobility (arm) in ad-hoc network routing. In *Conference on Network Protocols (ICNP'2001)*, Riverside, USA, November 2001. IEEE.

[2] Ian F. Akyildiz, James I. Pelech, and Bülent Yener. A virtual topology based routing protocol for multihop dynamic wireless networks. *Wireless Networks*, 7(4):413–424, July 2001.

[3] K. M. Alzoubi, P.-J. Wan, and O. Frieder. Distributed heuristics for connected dominating set in wireless ad hoc networks. *IEEE ComSoc/KICS Journal of Communications and Networks, Special Issue on Innovations in Ad Hoc Mobile Pervasive Networks*, 4(1):22–29, march 2002.

[4] Alan Amis, Ravi Prakash, Thai Vuong, and Dung Huynh. Max-min d-cluster formation in wireless ad hoc networks. In *Proceedings of IEEE INFOCOM*, pages 32–41, Tel-Aviv, Israel, March 1999. IEEE.

[5] Suman Bannerjee and Samir Khuller. A clustering scheme for hierarchical control in wireless networks. In *Infocom*, Anchorage, Alaska, April 2001. IEEE.

[6] Lichun Bao and J.J. Garcia Luna Aceves. Topology management in ad hoc networks. In *Mobihoc*, Anapolis, USA, June 2003. ACM.

[7] Stefano Basagni, Damla Turgut, and Sajal K. Das. Mobility-adaptive protocols for managing large ad hoc networks. In *Proceedings of IEEE International Conference on Communications (ICC)*, pages 1539–1543, Helsinki, Finland, June 2001. IEEE.

[8] Prithwish Basu, Naved Khan, and Thomas Little. A mobility based metric for clustering in mobile ad hoc networks. In *Proceedings of Distributed Computing Systems Workshop 2001*, Phoenix, Arizona, USA, April 2001. IEEE.

[9] Ljubica Blazevic, Levente Buttyan, Srdan Capkun, Silvia Giordano, and Jean-Yves Le Boudec. Self-organization in mobile ad-hoc networks: the approach of terminodes. *IEEE Communications Magazine*, 39(6):166–174, June 2001.

[10] Ljubica Blazevic, Silvia Giordano, and Jean-Yves Le Boudec. Anchored path discovery in terminode routing. In *Proceedings of The Second IFIP-TC6 Networking Conference (Networking 2002)*, Pisa, Italy, May 2002. IFIP.

[11] Sergiy Butenko, Xiuzhen Cheng, Ding-Zhu Du, and Panos M. Pardalos. *On the construction of virtual backbone for ad hoc wireless networks*, volume 1 of *Cooperative Systems*, chapter 3, pages 43–54. Kluwer Academic Publishers, January 2003.

[12] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, August 2002.

[13] Mihaela Cardei, Xiaoyan Cheng, Xiuzhen Cheng, and Ding-Zhu Du. Connected domination in ad hoc wireless networks. In *Sixth International Conference on Computer Science and Informatics (CSI 2002)*, North Carolina, USA, March 2002.

[14] Marco Carli, Alessandro Neri, and Andrea Rem Picci. Mobile ip and cellular ip integration for inter access networks handoff. In *International Conference on Communications (ICC'2001)*, Helsinki, Finland, June 2001. IEEE.

[15] Mainak Chatterjee, Sajal K. Das, and Damla Turgut. A weight based distributed clustering algorithm for mobile ad hoc networks. In *International Conference on High Performance Computing*, pages 511–521, Bangalore, India, December 2000. IEEE, ACM (sponsors).

[16] Mainak Chatterjee, Sajal K. Das, and Damla Turgut. Wca: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5(2):193–204, April 2002.

[17] Geng Chen, Fabian Garcia, Julio Solano, and Ivan Stojmenovic. Connectivity based k-hop clustering in wireless networks. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2002. IEEE.

[18] Yuanzhu Peter Chen and Arthur L. Liestman. Approximating minimum size weakly-connected dominating sets for clustering mobile ad hoc networks. In *Proceedings of the third ACM international symposium on Mobile ad hoc networking and computing*, pages 165–172, Lausanne, Switzerland, June 2002. ACM.

[19] Xiuzhen Cheng and Ding-Zhu Du. Virtual backbone-based routing in multihop ad hoc wireless networks. Technical Report 002, University of Minnesota, June 2002.

[20] Thomas Clausen, Philippe Jacquet, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized link state routing protocol. draft-ietf-manet-olsr-07.txt, November 2002.

[21] Daniel Câmara and Antonio Loureiro. A novel routing algorithm for ad hoc networks. In *33rd Hawaii International Conference on System Sciences (HICSS'2000)*, volume 8, Maui, Hawaii, January 2000. IEEE.

[22] Bevan Das, Raghupathy Sivakumar, and Vaduvur Bharghavan. Routing in ad-hoc networks using a spine. In *Sixth International Conference on Computer Communications and Networks (ICCCN '97)*, page 64, Las Vegas, USA, September 1997. IEEE.

[23] Thierry Ernst. *Network Mobility Support in IPv6.* Mathematics and computer science, University Joseph Fournier, Grenoble, France, 2001.

[24] Laura Marie Feeney. A taxonomy for routing protocols in mobile ad hoc networks. In *International Symposium on Handheld and Ubiquitous Computing (HUC'99)*, volume 8, Karlsruhe, Germany, September 1999. ACM.

[25] Yaacov Fernandess and Dahlia Malkhi. K-clustering in wireless ad hoc networks. In *Proceedings of the second ACM international workshop on Principles of mobile computing*, pages 31–37, Toulouse, France, October 2002. ACM Press.

[26] Fabian Garcia, Julio Solano, and Ivan Stojmenovic. Connectivity based k-hop clustering in wireless networks. *Telecommunication Systems*, 22(1):205–220, 2003.

[27] Sudipto Guha and Samir Khuller. Approximation algorithms for connected dominating sets. In *European Symposium on Algorithms*, pages 179–193, Barcelona, Spain, Septembre 1996.

[28] Zygmunt J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Conference on Universal Personal Communications (ICUPC'97)*, San Diego, USA, October 1997. IEEE.

[29] Zygmunt J. Haas and Marc R. Pearlman. The performance of query control schemes for the zone routing protocol. In *SIGCOMM'98*, Vancouver, Canada, September 1998. ACM.

[30] Zygmunt J. Haas and Marc R. Pearlman. Providing ad-hoc connectivity with the reconfigurable wireless networks. In *SIGCOMM'98*, pages 166–177, Vancouver, British Columbia, Canada, September 1998. ACM.

[31] Ting-Chao Hou and Tzu-Jane Tsai. An access-based clustering protocol for multihop wireless ad hoc networks. *IEEE Journal on Selected Areas in Commnications*, 19(7):1201–1210, July 2001.

[32] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[33] P. Krishna, Nitin Vaidya, Mainak Chatterjee, and Dhiraj Pradhan. A cluster-based approach for routing in dynamic networks. In *SIGCOMM Computer Communication*, pages 49–65, Cannes, France, April 1997. ACM.

[34] Taek Jin Kwon and Mario Gerla. Efficient flooding with passive clustering (pc) in ad-hoc networks. *ACM SIGCOMM Computer Communication Review*, 32(1):44–56, January 2002.

[35] B. Liang and Z. J. Haas. Virtual backbone generation and maintenance in ad hoc network mobility management. In *INFOCOM'2000*, Tel-Aviv, Israel, March 2000. IEEE.

[36] Chunhung Richard Lin and Mario Gerla. Adaptive clustering for mobile wireless networks. *IEEE Journal of Selected Areas in Communications*, 15(7):1265–1275, 1997.

[37] Madhav V. Marathe, Heinz Breu, Harry B. Hunt III, S. S. Ravi, and Daniel J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, December 1995.

[38] Shree Murthy and J.J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *ACM Mobile Networks and Applications Journal*, 1(2):18–197, October 1996.

[39] S.Y. Ni, Y.C. Tseng, Y.S. Chen, and J.P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Mobile Computing and Networking (MobiCom'99)*, pages 151–162, Seattle, USA, August 1999. ACM.

[40] Marc R. Pearlman and Zygmunt J. Haas. Determining the optimal configuration of the zone routing protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1395–1414, June 1999.

[41] Charles E. Perkins. *Ad hoc networking*, addison-wesley edition, 2001.

[42] Charles E. Perkins, Elisabeth M. Belding Royer, and Samir R. Das. Ad hoc on demand distance vector routing. draft-ietf-manet-dsr-13.txt, February 2003.

[43] Izhak Rubin, Xiaolong Huang, Y. C. Liu, and Huei-jiun Ju. A distributed stable backbone maintenance protocol for ad hoc wireless networks. In *VTC*, Jeju, Korea, April 2003. IEEE.

[44] Bo Ryu, Jackson Erickson, Jim Smallcomb, and Son Dao. Virtual wire for managing virtual dynamic backbone in wireless ad hoc networks. In *3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, Seattle, USA, August 1999. IEEE.

[45] Ahmed Safwat and Hossam Hassanein. Infrastructure-based routing in wireless mobile ad hoc networks. *The Journal of Computer Communications*, 25(3):210–224, 2002.

[46] Cesar A. Santivanez, Bruce McDonald, Ioannis Stavrakakis, and Ram Ramanathan. On the scalability of ad hoc routing protocols. In *INFOCOM'02*, New York, USA, June 2002. IEEE.

[47] Prasun Sinha, Raghupathy Sivakumar, and Vaduvur Bharghavan. Cedar: a core-extraction distributed ad hoc routing algorithm. In *INFOCOM'99*, pages 202–209, New York, USA, March 1999. IEEE.

[48] Prasun Sinha, Raghupathy Sivakumar, and Bharghavan Vaduvur. Enhancing ad hoc routing with dynamic virtual infrastructures. In *INFOCOM'01*, Anchorage, Alaska, USA, April 2001. IEEE.

[49] Raghupathy Sivakumar, Bevan Das, and Vaduvur Bharghavan. The clade vertebrata: spines and routing in ad hoc networks. In *International Symposium on Computer Communications (ISCC'98)*, pages 599–605, Athens, Greece, June 1998. IEEE.

[50] Raghupathy Sivakumar, Bevan Das, and Vaduvur Bharghavan. Spine routing in ad hoc networks. *Cluster Computing*, 1(2):237–248, 1998.

[51] Raghupathy Sivakumar, Prasun Sinha, and Vaduvur Bharghavan. Braving the broadcast storm: Infrastructural support for ad-hoc networks. *International Journal on Computer Telecommunications Networking*, 41(6):687–706, April 2003.

[52] Saurabh Srivastava and R. K. Ghosh. Cluster based routing using a k-tree core backbone for mobile ad hoc networks. In *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 14–23. ACM Press, 2002.

[53] Ivan Stojmenovic, Mahtab Seddigh, and Jovisa Zunic. Dominating sets and neighbor elimination-based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 12(12):14–25, December 2001.

[54] John Sucec and Ivan Marsic. Clustering overhead for hierarchical routing in mobile wireless networks. In *INFOCOM*, pages 202–209, New York, USA, June 2002. IEEE.

[55] Robert Vilzmann. Analysis of different clustering techniques for mobile ad hoc networks. Technical Report EE381K.11, University of Texas at Austin, 2002.

[56] Jie Wu, Ming Gao, and Ivan Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. In *International Conference on Parallel Processing (ICPP '01)*, Valencia, Spain, September 2001. International Association for Computers and Communications.

[57] Shih-Lin Wu, Sze-Yao Ni, Yu-Chee Tseng, and Jang-Ping Sheu. Route maintenance in a wireless mobile ad hoc network. In *33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000. IEEE.

[58] Kaixin Xu, Xiaoyan Hong, and Mario Gerla. Landmark routing in ad hoc networks with mobile backbones. *Journal of Parallel and Distributed Computing*, 63(2):110–122, 2003.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399